Applying deep learning and synthetic data to improve power line inspection

Master's Thesis in Computer Science

Le Tuan Tai

June 15, 2020
Halden, Norway

Abstract

As electricity has played a critical role in the society, preventive maintenance and inspection for power line objects become essential factors in preserving the availability of power transmission.

Deep Learning in power line inspection is a technological shift that helps to reduce the chance of missing faults, which usually occurs during human's inspection. Because deep learning requires a massive amount of training samples to learn effectively, the literature suggested synthetic data to fill in the lack of real-world samples. However, there are still a challenge of so-called "synthetic gap" between the synthetic domain and the real-world domain, which can negatively affect the application of deep learning models.

In this thesis, we focus on the worn chain shackles regression, a case study of power line inspection at eSmart Systems AS. By introducing transfer learning, we have reduced the current solution loss by approximately six percentage points, from 0.0717 to 0.0077. Besides, thanks to transfer learning with the state-of-the-art deep network architectures, we can train more precise models without using a margin in L1 loss function to compromise the loss. Furthermore, we use the insightful analysis of deep neural networks to adjust 3D variations, which reduce the hardship of picking 3D variables arbitrarily and searching exhaustively for the best combination. To the best of our own knowledge, we are pioneers that address the synthetic gap in preventive maintenance by investigating the learned experience of the convolutional neural networks. Furthermore, after performing experiments with the ResNet50, the VGG16, and the MobileNet architectures, we introduce the rule-of-thumb training techniques for each network architecture to maximize the benefits of synthetic data at best.

Keywords: Deep Learning, synthetic data, convolutional neural networks, synthetic gap.

Acknowledgments

I want to express my sincere thank to Prof. Roland Olsson. Under his supervision and guidance, I have discovered my ability to push further my limitation and master the knowledge of Machine Learning.

I would not have finished this thesis without the support of eSmart Systems AS. A special tribute to Dr. Davide Roverso, Chief of Analytics Department, for offering me an internship at eSmart Systems. My sincere thanks to Dr. Nguyen Van Nhan, my direct supervisor at eSmart, who has helped me define the thesis's problem and provided me with the initial solutions developed at eSmart which I use as the baseline in this thesis. I appreciate the inspiring discussions and stress-free small talks I had with Huynh Chi Hieu and Nguyen Danh Tuan.

I would like to thank my family, my grandmother, and my grandaunt for encouraging me and love me unreservedly. I would like to thank my girlfriend, Iris Phuong Thao, for her non-stop supports and unconditional love. Besides, I would like to thank my friends and my second family in Norway: Tai Huynh and Dung Ngo, for their everlasting friendships and unforgettable journeys.

Prerequisites

Because this thesis spreads over many different aspects of machine learning and deep learning, it is not possible for the author to go into every detail of all the subjects that are mentioned. Consequently, it is assumed that readers already have knowledge of machine learning at introduction level. Besides, readers should have basic knowledge of artificial neural network and its learning procedure, for example, feed forwarding, back propagation. He or she should also be familiar with basic calculus such as derivative, integration.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and motivation

Since being introduced in 1887, electricity has played an important role in modern society. A power failure can cost business and public services financially a decent amount, not to mention humans lives. For example, in the U.S., due to "fretting erosion" "a worn C-hook on Tower No.27-222 on the nearly 100-year-old line that failed and caused the 115,000-volt line to hit the tower and spark the fire that destroyed the town of Paradise and left 85 people dead" (Derbeken, 2019). Therefore, it is critical to conduct preventive maintenance to maintain the reliability, availability, and sustainability of the power-transmitting system.

Preventive maintenance is the maintenance conducted regularly on a piece of equipment to reduce the probability of that equipment failing (Hinds, 1985). Recently, the maintenance has shifted from human supervision to computer-assisted methods, for example, applying machine learning model on power grid (Wilson, Coyle, & Thomas, 2017), water pumpers (Lee et al., 2019) computer vision to detect vibrations issues with the equipment (Klosterman, 2016)

Power line inspection, which is the prior task in the preventive maintenance for electricity objects, has also followed this technical shift. Traditionally, the inspection is usually done either by foot patrol or by helicopter-assisted methods. Image data will be collected and analyzed for faults by humans' vision. This task is expensive, tedious, dangerous, and contains a high risk of missing culprits. (Nguyen, Jenssen, & Roverso, 2018). In the fire accident mentioned above, inspectors had missed the worn C-hook during aerial and ground patrols.

There is a need for a new inspection method, a vision-based inspection of the power line with the support of technology. Image analysis can be done automatically with humans' verification thanks to the development of deep learning and computer vision. This advancement places a foundation for an autonomous power inspection shortly.

It is necessary to train the deep learning models on an extensive amount of labeled data to maximize deep learning's generalization. Unfortunately, the process of collecting training data is labor-consuming and sensitive to humans' faults (Nguyen et al., 2018). Some power line components either appear too small in the image or hardly come up with an adequate number of samples in the image repository. Fortunately, (Nguyen et al., 2018) suggested that synthetic image data can be the solution for training data shortage and class imbalance problems.

eSmart Systems AS, a software solutions company, has developed many AI-powered solutions for inspecting power line objects. The most common challenge eSmart is facing is the lack of real-world training images. For example, some power line faults such as missing top pads, wood-peckers, worn metal equipment, are oddly found in the real-world inspection

(Nguyen et al., 2018). To overcome this shortage, eSmart has brought synthetic data into practice to develop preventive maintenance solutions for power line faults.

Even though synthetic data is available, combining them with deep learning to generate a sound solution for power line inspection could be another challenge. Firstly, there is a trade-off between the fidelity and the time for synthesizing data. Put differently, low-fidelity simulators save the generating time. However, they can not reproduce the noise and richness of real-world objects. On the contrary, photo-realistic images require a careful and time-consuming imitation of physical relations. The difference between synthetic data and real-world data is called *reality gap* or domain gap. Hence, there is a risk that the synthetic data can cover a domain with little or no conjunction with the real-world domain. Therefore, a second challenge comes as a consequence: deep learning models trained on synthetic data might be prone to overfit on training samples. They show very little or no use on real-world images or even synthetic images generated with the same settings. We would iterate in this details in section 3.4

1.2 Research question

We raise this research question to address the challenges mentioned above:

RQ How to bring deep learning and synthetic data into practice to improve a power line inspection task?

In order to thoroughly answer the research question, we raise below sub-questions:

SQ1 What are deep learning architectures that can utilize synthetic data at best?

SQ2 How do the insights of deep neural networks to can reduce the negative effect of *reality gap*?

SQ3 Which are the rule-of-thumb deep learning practices that can maximize the benefit of synthetic data?

1.3 Scope

There are two conventional strategies to bridge the synthetic gap: increasing the resemblance between two domains or exploring methods that can train models on a more generic domain (Borrego et al., 2018). In the first strategy, the literature attempts to increase the fidelity of simulators to generate more photo-realistic outputs. Another approach is using GANs (Generative Adversarial Networks) to make synthetic images more photo-realistic. Unfortunately, both require a significant effort to create simulators that can represent complicated physics' phenomena.

In this thesis, we would like to use network visualization to gain a better understanding of the synthetic gap between the real-world domain and synthetic domain. Later, this knowledge can help us to tune the simulator's settings efficiently. In other words, we do not consider the exhaustive search over combinations of 3D variables or training GANs as our approach.

1.4 Report outline

We organize this thesis into seven chapters: chapter 1 briefly summarizes the existing challenges of vision-based power line inspection and raises the research question of our thesis. Chapter 2 provides readers with a fundamental understanding of terminology and terms that we use in this report. Chapter 3 reviews the application of synthetic data on different deep learning problems and briefly introduces the case study at eSmart Systems AS. After conceiving a comprehensive view of the problem, we project our academic objectives and propose the experimental setup to address the vision-based power line inspection in chapter 4. We provide the results in chapter 5 and discuss them in-depth in chapter 6. Last but not least, we conclude our findings and answer the research question posed in the first chapter as well as brainstorm future developments of this thesis.

Chapter 2

Background and Theory

This chapter will provide readers with the necessary information to understand the background and theory that we have used in this thesis. In section 2.1, we will cover basic concepts of machine learning and training machine learning models. Section 2.2 explains in depth the concept of deep learning with its backbone convolutional neural networks. Readers who are already familiar with machine learning and deep learning can skip this chapter.

2.1 Machine Learning

As (Mitchell, 1997) explained, Machine Learning is "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." Therefore, each machine learning problem can be defined as the problem of improving some measure of performance P when executing some task T, through some training experience E.

In other words, given task T, gaining experience E is a learning procedure that extracts knowledge on seen data and infers pattern(s) that generalize on future, unseen data. In order to evaluate that "generalization ability", a metric P should be introduced beforehand.

Each machine learning algorithm can be categorized as *supervised* or *unsupervised* learning. In *supervised learning*, the training data includes both observations and expected outputs. Usually, metric P, a.k.a loss function, calculates the disparity between algorithm's predictions and the truth of the observations. On the other hand, unsupervised learning refers to training with unlabeled data. The algorithm is required to learn the patterns of data without any supervisors. (Goodfellow, Bengio, & Courville, 2016)

2.1.1 Artificial Neural Networks

Perceptron

A perceptron, a.k.a neuron, takes a vector X contains $x^{(1)}, x^{(2)}, ..., x^{(n)}$ as input and linearly weights it with an offset bias.

$$z = a(x) = \sum_{i=1}^{n} w^{(i)} x^{(i)} + b \tag{2.1}$$

Later, an activation function $g(t)$ is applied at the output z of the neuron

$$g(z) = g(a(x)) \tag{2.2}$$

Artificial Neural Networks

ANNs are inspired by the neural system of the human brain system. An ANN consists of layers stacking on each other. A layer is constructed of neurons (a.k.a perceptrons, nodes). Although the depth and breadth of ANNs might vary, the common architecture contains one input layer, one output layer, and at least one hidden layer, as depicted in Figure 2.1. The output of the previous layer will be the input of the next layer (Goodfellow et al., 2016).

The output layer is designed based on the task T of the neural network.

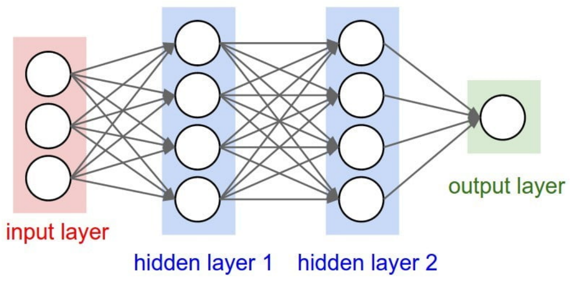


Figure 2.1: An example of ANN with two hidden layers (image from Stanford CS231 course)

2.1.2 Gradient-based learning

The learning procedure of a neural network is the process of optimizing the weights in order to reduce the loss L measured by metric P.

It is a repetitive approach with three step:

- Step 1: Feed-forward pass, the network takes an input x and forward it through until it reaches the output layer.
- Step 2: Backpropagation: a cost will be calculated based on the loss function L then passed backward until it reaches the first hidden layer, to compute the gradient
- Step 3: Weight update: the network weights will be adjusted as follow

$$W = W - \alpha \frac{\partial L}{\partial W} \tag{2.3}$$

where α is the learning rate.

2.1.3 Challenges of gradient-based learning

Generalization and bias-variance trade-off

Generalization refers to how well the experience learned by a machine learning model applies to unseen data. A lower error on this test data implies higher generalization and vice versa (Goodfellow et al., 2016).

Overfitting indicates a model performs too well on the training data but predicts poorly on new data. Overfitting happens when a model picks both the patterns of training data and the noise to learn. Consequently, when the noise pattern of the training data does not exist on test data, the model tends to predict with low accuracy. On the other hand, when a model performs poorly both on the training set and test set, it is underfitting. It usually happens when the problem domain is too complicated, and the model lacks the learning capacity.

The bias-variance trade-off can explain the ability to generalize on unseen data of any machine learning algorithms. A model with high bias will underfit to the training data, resulting in high training error rates. High bias models are limited in what they can learn and are restricted to a more narrow problem domain then what the training task requires. On the contrary, a high variance model is sensitive to noise and random patterns on train data. As a result, it will often overfit to training data, show low error rates during training, while generalize poorly on the test data (Goodfellow et al., 2016).

The bias-variance trade-off is a fundamental concept within machine learning. Reducing the variance of a model will result in an increase in its bias, and vice versa. To optimize performance in a model, it is critical to find a balance between bias and variance with the help of regularization. Regularization techniques add certain constraints to a high-variance model to lower its variance (Bishop, 2006).

Vanishing gradients and exploding gradients

Vanishing gradients happens when the gradient becomes to small that the learning procedure unknowingly where to go to optimize the loss. More mathematically explained, the derivative of the loss function is close to 0.

Exploding gradient, on the other hand, happens when the gradient signals become too big and make the learning unstable.

Local optima

The solution of the neural networks is non-convex and rigid, which means the gradient can be pointing towards local optima. Therefore, gradient descent is not guaranteed to reach the global minimum of the loss function.

Adjusting hyper-parameters

Hyper-parameters are the parameters that are set before the learning procedure happens. For example, the learning rate α, the mini batch size are hyper-parameters which are defined before the training takes place.

Setting hyper-parameters is challenging, because they are highly dependent on a specific dataset and vary among different machine learning problems. Besides, a single hyper-parameter can have a significant impact on the training process. Take learning rate into consideration, if the learning rate is too big, it causes overshooting, in which the solution is goes far over the global optima. On the other hand, if the learning rate is too small, it can slows learning process.

2.1.4 Regularization: Solutions for those challenges

Batch Normalization

In the training process of neural networks, the parameters (weights and biases) are updated frequently after each batch. This causes the distributions of internal layers to change every time the network parameters are updated, which is called Internal Co-variate Shift.

These changes slow down the learning process of the networks because the learning algorithm is chasing different objectives every time a new batch is fed in. Therefore, to speed up the learning process, batch normalization normalizes output activation between layers.

Batch normalization can be implemented during training by calculating the mean μ and variance σ^2 of each input variable to a layer per mini-batch containing m samples. Assume Z as the output of one layer, Z contains $z^{(1)}, z^{(2)}, ..., z^{(m)}$.

$$\mu = \frac{1}{m}\sum_{i=1}^{m} z^{(i)} \tag{2.4}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(z^{(i)} - \mu) \tag{2.5}$$

After that, these statistics are used to normalize each output $z^{(i)}$.

$$z^{(i)}_{norm} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{2.6}$$

where ϵ is an offset factor in case the variance σ^2 is zero.

Finally, batch normalization technique scales and shifts is conducted as below:

$$\hat{z}^{(i)} = \gamma * z^{(i)}_{norm} + \beta \tag{2.7}$$

γ, β are learn-able parameters.

Dropout

Dropout is a technique where randomly selected neurons are ignored. Therefore, their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014).

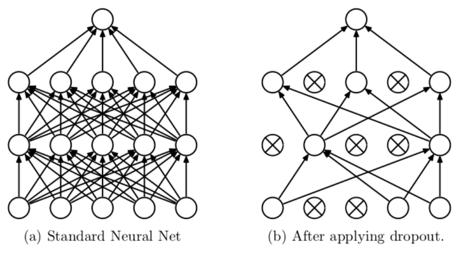


Figure 2.2: Illustrations of a standard ANN (a) and a ANN with dropout (b) (image from (Srivastava et al. 2014))

If an ANN contains many fully connected layers, it is prone to overfitting. One method to reduce overfitting is dropout. The training phase happens on a simplified version of the original network, where individual nodes are either dropped out of the net with probability (1-p) or kept with probability p. Incoming and outgoing edges to a dropped-out node are also removed. Only the reduced network is trained on the data in that stage. The removed nodes are then reinserted into the network with their original weights.

Before the weights are finalized, the units which are retained from the network during training will be multiplied by p during the test time. This weights scaling can be performed at training time instead, after updating the weights every mini-batch is fed forward. Consequently, the output at test time will be left unchanged (Srivastava et al., 2014).

Adam optimizer

Adam is optimization algorithm which keeps a moving average over past gradients (Kingma & Ba, 2017) . Also, it implemented adaptive learning rates for each individual weight.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{S}_t + \epsilon}} \hat{V}_t \tag{2.8}$$

where

$$\hat{V}_t = \frac{V_t}{1 - \beta_1^t} \tag{2.9}$$

$$\hat{B}_t = \frac{S_t}{1 - \beta_2^t} \tag{2.10}$$

$$V_t = \beta_1 V_t - 1 + (1 - \beta_1) \frac{\partial L}{\partial w_t} \tag{2.11}$$

$$S_t = \beta_2 V_t - 1 + (1 - \beta_2) (\frac{\partial L}{\partial w_t})^2 \tag{2.12}$$

α is the learning rate, ϵ is a small value that prevents division by zero, β_1, β_2 adjust the two moving averages.

2.2 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a designated network for grid-like input, for example, images.

2.2.1 Convolution operator

Convolution is a function derived from two given functions by integration which expresses how the shape of one is modified by the other.

$$s(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau \tag{2.13}$$

In convolution term, the first function f is referred as *input* and the second function g is the *kernel*. The output s is called *feature map*. In that context, the convolution formula can be described as a weighted average of the function $f(\tau)$ at the moment t where the weighting is given by $g(-\tau)$ simply shifted by amount t. As t changes, the weighting function emphasizes different parts of the input function (Smith, 1997).

In the application of computing, the continuous signal is impossible, for example, the signal is measured once a second. Therefore the time t can be represented as discrete values. Equation 2.13 can be discretized as follow:

$$s(t) = (f * g)(t) = \sum_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau \tag{2.14}$$

2.2.2 Pooling layer

A pooling layer (max or mean function) applies over regions of the feature maps. This layer helps reduce the overall resolution of the feature maps. Reducing the dimension of the output helps reduce the risk of overfitting (Goodfellow et al., 2016).

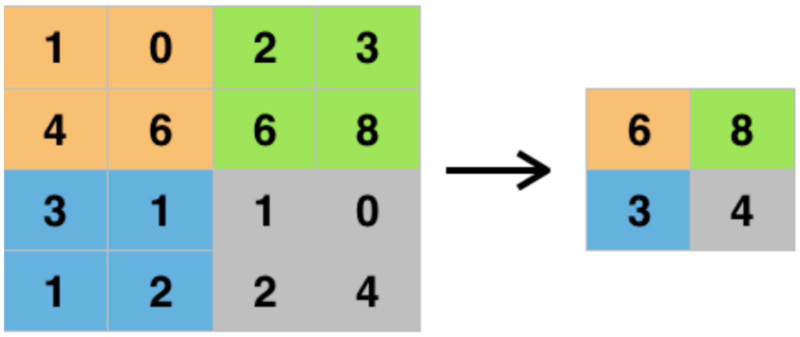


Figure 2.3: An example of max pooling on a 2D input (image from DeepAI)

2.2.3 CNN and Image Processing

A CNN is a stack of many convolutional layers followed by fully connected layer(s).

Usually, the CNN is applied in image processing with multidimensional input, which requires to apply the convolution operations on many dimensions at the same time. Take into consideration of an image I that has $w*h*d$ dimensions and a kernel with $f*f*d$ dimensions. The output S at row i, column j is depicted below

$$S(i,j) = (I*K)(i,j) = \sum_{k=0}^{d}\sum_{m=0}^{f}\sum_{n=0}^{f} I(i+m, j+n, k)K(m,n,k) \tag{2.15}$$

where $i \in [0, \frac{w-f}{s}]$ and $j \in [0, \frac{h-f}{s}]$, s is the stride

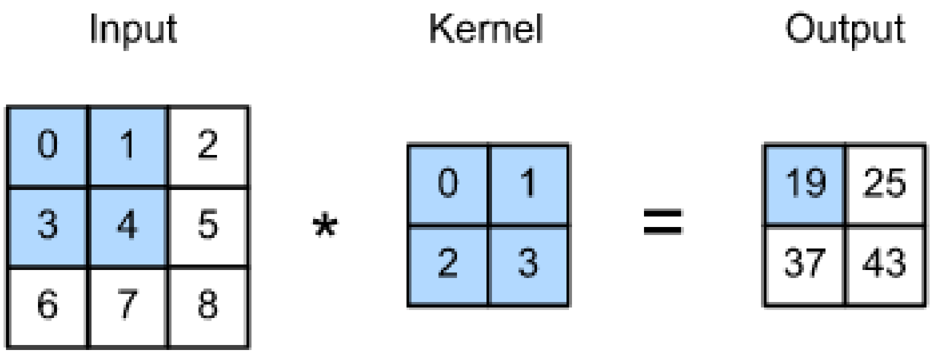


Figure 2.4: An example of convolution on grey image (image from d2l.ai)

A convolutional layer consists of kernels, an activation function, and a pooling layer:

- Kernels are a set of linear functions to apply to the input to extract features. The output of applying the kernels are called feature-maps.

- The activation function is a non-linear function, such as ReLU, that is applied to the feature maps.
- A pooling layer to reduce the overall resolution of the feature maps.

2.2.4 Depthwise separable convolution

In a normal convolution, a kernel, which has the same depth as the input, slides along the input to produce a single 2-D feature map of size $m * n$. Put differently, on an image of size $w * h * d$, to produce k feature maps of size $m * n * k$, we need k kernels. Consequently, it can make a lot of multiplications, which is challenging for computation. This computation burden can be solve by a depthwise separable convolution . A depthwise separable convolution includes two parts: a depthwise convolution and a pointwise convolution.

Depthwise convolution applies convolution and reserves the depth d of the input image size $w * h * d$. Therefore, it has d kernel. The size of the kernel will be calculated to produce the output size of m*n*d, which later be forwarded to pointwise convolution.

Secondly, the pointwise convolution uses kernels of size $1 * 1 * d$ to iterate thought every single point of the $m * n * d$ feature maps. Hence, k kernels should be utilized to produce the expected feature maps of size $m * n * k$.

2.2.5 CNN architecture

Since 2012, when first CNN architecture, AlexNet, won top-5 error in ImageNet Large Scale Visual Recognition Challenge (ILSVRC), CNNs have reserved their championship throughout years. In order to prove the superiority of the architecture, people usually refer to the ILSVRC dataset as a benchmark.

VGG

VGG Net won the second place in ILSVRC 2014, (Simonyan, Vedaldi, & Zisserman, 2014), with 7.3% of top-5 error rate on ILSVRC challenge.

VGG-16 is composed of 16 layers compact in 5 blocks, each block is followed by max pooling layer. Later, they are flatten with 3 fully-connected layers.

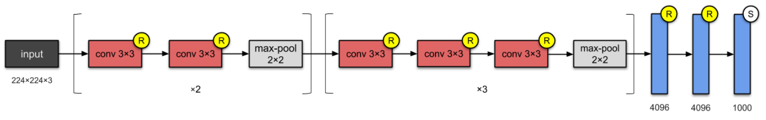
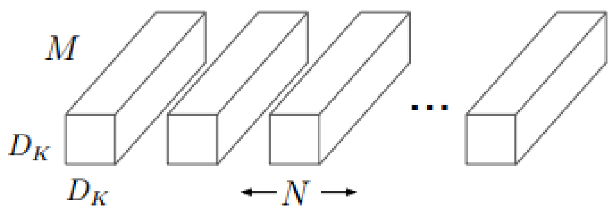


Figure 2.5: VGG16 architecture (from Raimi Karim Medium)

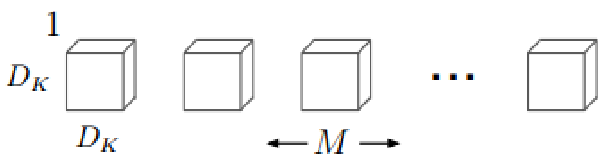
MobileNet

MobileNet was a deep neural network architecture designated to mobile devices. MobileNets use depthwise separable convolutions (figure 2.6) to build light-weight deep neural networks (Howard et al., 2017).

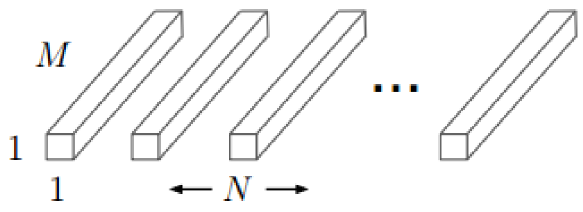
MobileNets still utilize standard convolution at the few first layers. Other layers apply depthwise separable convolution instead.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2.6: Comparision between standard convolution (a) and depthwise seperable convolutions (b,c)

ResNet

In Deep Learning, stacking more convolutional layers is believed to better the performance of deep learning models. However, when the network grows deeper, the network suffers from vanishing gradient. Therefore, Residual Networks (ResNet) (He, Zhang, Ren, & Sun, 2016) from Microsoft was introduced. The idea of ResNet is based on residual blocks, which allow the activation of a layer to skip one or several layers, as depicted below (figure 2.7)

This solution achieved 3.6% top-5 error on the ILSVRC 2015.

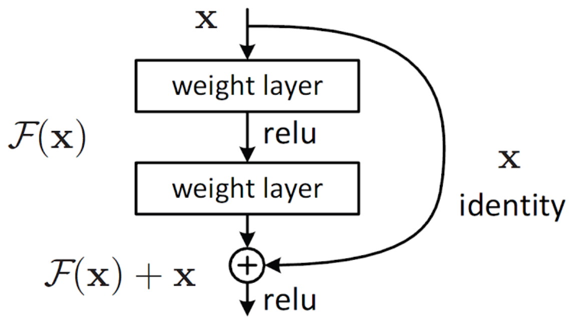


Figure 2.7: A residual block of ResNet(image from (He et al., 2016))

Chapter 3

Related Work

This chapter will cover the literature related to the research question raised in chapter 1. Section 3.1 covers the related work on the application of synthetic data and deep learning. Section 3.2 provides the readers with a brief introduction of CNNs visualization to understand the learned patterns of a trained deep learning model. Section 3.3 summarizes the comprehensive study of vanilla deep regression, a deep neural network designated to regression problem. Section 3.4 outlines the "worn chain shackles" case study, an industrial power line inspection task at eSmart Systems, which inspired the further investigation in this thesis.

3.1 Synthetic data and Deep Learning

Synthetic data refers to data that is generated for a specific learning task (Lu et al., 2019), which is lack of observations. Synthetic data can be generated either by doing manual data modeling or by using video game engines (Mayer et al., 2018). Synthetic data have been broadly applied to various tasks in Deep Learning. For example:

- *object detection* for cars (Tremblay et al., 2018), indoor scenes (Dwibedi, Misra, & Hebert, 2017) (Georgakis, Mousavian, Berg, & Kosecka, 2017), robotic grasping (Tobin et al., 2017)
- *object classification* for military vehicles (Lundmark, 2019), car type (Movshovitz-Attias, Kanade, & Sheikh, 2016), medical fMRI data (Zhuang, Schwing, & Koyejo, 2019)
- *object segmentation* for humans' postures (Varol et al., 2017)
- other tasks in deep learning, for examples, 3D face reconstruction (Richardson, Sela, & Kimmel, 2016), learning disparity and optical flow (Mayer et al., 2018)

As mentioned in the 1.2, there is a difference between the synthetic data and real-world data called reality gap. To close the reality gap, there are two common approaches: either making the synthetic domain closer to real-world domain or seeking training methods that can create a model representing both domains. The former approach can be implemented by generating photo-realistic images using high-fidelity simulators or using a Generative Adversarial Networks (GANs) to improve the photo-realism of the images. Both approaches are tedious, time-consuming and hard-labor work.

3.1.1 Object detection

Object detection benefits most from the application of synthetic data because the object labeling usually is automatically generated during the data synthesize process.

(Borrego et al., 2018) trained a simple object detector for three object shapes: sphere, box, cylinder. MobileNet, pre-trained on ImageNet dataset, was fine-tuned on synthetic images followed by real-world images. They conducted two ablation studies: camera viewpoint (fixed and randomly generated) and different textures complexity. However, because the classes of this research are not affected by most of data augmentation techniques, this article should have applied more data augmentation techniques rather than only simple horizontal flipping and random cropping. The authors reported that model which was trained on fixed camera viewpoint dataset and and fine-tuned on real-world images had the best performance. Besides, the article found that increasing texture complexity should associate with an increasing number of training samples.

(Dwibedi et al., 2017) trained an object detector for simple objects: cereal boxes and coffee cups. To curate the dataset, the authors cut and pasted target objects on different backgrounds. As a literature overlap, (Georgakis et al., 2017) curated synthetic data by similar cutting and pasting techniques. This data collection process is fast and time-saving compared to using simulators. Besides, it benefits from using the objects from real-world domain. However, all perspectives of a target object should be taken. Besides, the model hardly experiences the variance of other factors, e.g, lighting condition, camera angle, because those factors are constant values when the objects are cut from the scene.

(Hinterstoisser, Lepetit, Wohlhart, & Konolige, 2018) trained an object detector on different network architectures (Faster-RCNN, RFCN, Mask RCNN). The authors generated synthetic data by adding Gaussian noise to the object of interest and Gaussian blurring the object edges before composing over a background image. The networks were trained on real images first. Later, feature extracting layers were frozen, while the remaining layers were trained on *synthetic images only*. The paper concluded that the pre-trained features extractors are already rich enough and do not need to be retrained. Also, simple rendering techniques improve the performance of the detector.

(Tobin et al., 2017) trained a mechanical arm to grasp the objects in the real world by using the synthetic data rendered from 3D models. He did a careful ablation study of each domain factors, e.g, noise, camera randomization, non-related objects, a.k.a, distractors. The paper paid attention to the different colors and textures of the target object, background images, number of lights in the scenes, the pose of the lights, camera position. Only synthetic data was used for training. The output model was used for testing without further training on real images. The object detector is reported to achieve high accuracy to perform grasping clutter. However, the shapes of objects and the scenarios (background complexity) in this research were simple.

(Tremblay et al., 2018) applied the technique of domain randomization on the problem of car detection. The author did a careful ablation study of factors, light, texture, data augmentation, flying distractors. Synthetic data with more variance and non-realistic characteristics was claimed to have the best outcome. The article also found that the idea of frozen feature extractors from (Hinterstoisser et al., 2018) show less impact work for their problem because Tremblay's target classes are more varied. The author also concluded the model trained with synthetic data and fine-tuned with real images later outperformed the one with only real images. However, the authors ignored the importance of the background in the study.

(Prakash et al., 2018) applied the SDR (Structured Domain Randomization) into car detection. The paper paid attention to the context is a camera assembled to a car in traffic. Several constraints were applied for synthetic data, for example, number of trajectory, the number of cars on the street, the texture of cars, lighting, and weather. Training the models with synthetic data followed by real images was reported to produce state-of-the-art performance. However, this approach is not applicable to problems in which the constraints, e.g distance between objects and number of objects, are unclear.

3.1.2 Classification

(Lundmark, 2019) did a thorough thesis of the synthetic meta-learning for object classification. He concluded that the synthetic gap is more significant for 5-way 1-shot learning. Also, the lighting randomization during training can increase the accuracy by 8 percent point. He concluded that models trained with only synthetic data can perform pretty close to the model trained only on real images.

3.2 Visualization of Convolutional Neural Networks

Neural networks are considered as a black-box approach in machine learning algorithms. However, because convolutional neural networks mainly work with images, the output of each layer after applying the kernels (a.k.a. feature maps) is more conceivable to humans. Therefore, the learning process of CNNs can be visualized via the feature maps for debugging the training process.

3.2.1 Activation visualization

The term is straightforward, it means visualizing the activations of the network during the forward pass (Simonyan et al., 2014). From this inspection, if there are dead filters indicated by black activation outputs. it might suggest that the learning rates is pretty high.

3.2.2 Conv/FC Filters

Visualize the weights of the filters. There are two scenarios that can happen:

- Nice and smooth filters without noisy patterns indicate well-trained networks.
- Noisy patterns indicate that a network may not be trained long enough, or a sign of overfitting caused by very low regularization.

3.3 Deep Regression

As the application of deep learning in regression is limited, (Lathuilière, Mesejo, Alameda-Pineda, & Horaud, 2019) did a comprehensive study about the application of deep learning network on the regression. This is the first systematic review with a firm statistical foundation regarding vanilla deep regression, a.k.a deep neural architecture with a regression output. The authors chose four datasets for their study:

- Head-pose dataset, Biwi

- Facial landmark detection, FLD
- Human pose estimation, Parse
- Another human pose estimation, MPII

This paper studied two network architectures which were originally design for 1000-class classification problem on ImageNet dataset: VGG-16 and ResNet-50 as described in section 2.2.5

The authors analyzed the impacts of hyper parameters to the performance of network architecture on datasets. They categorized them into two groups: network optimizations (batch size, optimizers) and network variants (loss, batch normalization, layer normalization, dropout, fine-tuning depth, regressed layer).

3.3.1 Network optimization

The paper implemented different optimizers: AdaGrad, RMSProp, Adadelta, Adam. The paper suggested to use Adam and AdaDelta at a new task. Regarding the batch size, on VGG-16 and RestNet-50, the authors tried 16, 32, and 64 batches. Because of the memory limit, the batch size of 128 was tested only on VGG-16. ResNet-50 was reported to be independent from the batch size while VGG-16 performed better when the batch increased in size.

3.3.2 Network variants

The architectures were compiled on different loss functions: mean squared error (MSE), mean absolute error (MAE), and the huber loss (HUB). The results were inconclusive, which showed each loss function might be dependent both on the dataset and the network architecture.

Batch Normalization (BN) and Layer Normalization (LN). On VGG-16, the batch normalization was either set before the regression output (after the activations of fully connected layer FC2) or before the fully connected layer FC2. On ResNet-50, the normalization was set before the regression node. ResNet-50's performance showed no statistically significant difference after inserting BN. On the other hand, the performance on VGG-16 was reported to achieve a significant differecne by either adding BN or LN.

Because VGG-16 has two fully connected layers before the regression output, dropout was applied only on VGG-16 as a regularization. Intuitively, because ResNet-50 has no fully connected layer, drop out was not a concern. Dropout ratio lying in range [40% - 60%] proved to add a significant advantage for VGG-16 architecture.

Regarding fine-tuning depth, i.e. how many Convolutional blocks with pre-trained weights should be tuned. On VGG-16 architecture, all layers were suggested to be retrained, which means all five convolutional blocks. However, on ResNet-50, the first three or four blocks of the 5-block architecture should be tuned according to the new dataset, while tuning til the deepest block could caused a significant disadvantage to the vanilla deep regression.

To choose the regression layer, the output node was advised to set up after the activation of FC2 on VGG-16 architecture, On ResNet-50, the regression node put either after the 5th Convolutional block or the Global Average Pooling layer, was reported to add significant advantage to the performance.

3.4 Case study at eSmart Systems AS

At eSmart Systems, "Worn chain shackles" is to detect whether a joint between chain shackles wears off and and state how severe the worn joint is.

3.4.1 Challenges

There are several reasons that make worn chain shackles a challenging problem for vision-based power line inspection:

- The definition of "worn chain shackle" is non-existence and not universal between energy institutions.
- Decisions for replacement are made by experts, which is arbitrary.
- Images taken on the field of worn chain shackles usually are label-less
- Even when the data is given, human labeling worn surface percentage is an arbitrary process which is subject to the perception of the human. To achieve a high and reliable label, the voting technique, e.g. jury board with the attendance of trained experts, should be utilized. This is a labor-intensive and time-consuming process

To resolve these obstacles, the Analytics Team at eSmart introduced a regression deep learning model based on synthetic data. A trained regression model can be generalized to adapt to the different definitions of "worn chain shackle". Besides, the utility of synthetic data can avoid the controversial labeling problem because the worn percentage modeled precisely by the simulator based on the expected input of worn percentage.

3.4.2 Initial solution at eSmart

Synthetic data generation

This process is outsourced to Nordic Media Lab (NMLab) where they use Unreal Engine 4 (UE4) as a rendering engine. This allows them to create our own test world, where they control 3D variables and can easily alter the combinations to get the correct results.

In the first version, NMLab paid attention to lighting (environment light) and reflection on the surface. This contracted project is not ready to be published as a stand-alone system at the time of writing. Hence, it requires cooperation between the Analytics Department and NMLab each time new data is required.

Two chain shackle objects will be aligned with each other. The worn percentage labels, range from 0%-95%, are calculated based on the overlap section of two chain shackles.

They use collection of HDRI (high dynamic range imaging) images as background. The collection is a local folder in the project, and can be increased in number by adding more HDRI images. The backgrounds in the generator are based on a 360 image wrapped in a ball around the focus object. Figure 3.1 shows an example of a synthetic image generated in accordance with lighting condition and reflection constraints.

Texture: four textures are used base color, metallic, roughness and normal (figure 3.2), which will be applied on the surface by UV mapping technique.

Nordic Media Lab constructed four datasets, each dataset includes 20.000 images:

- NOTX_NOBG: the simplest dataset. There is no background, and no texture applied.

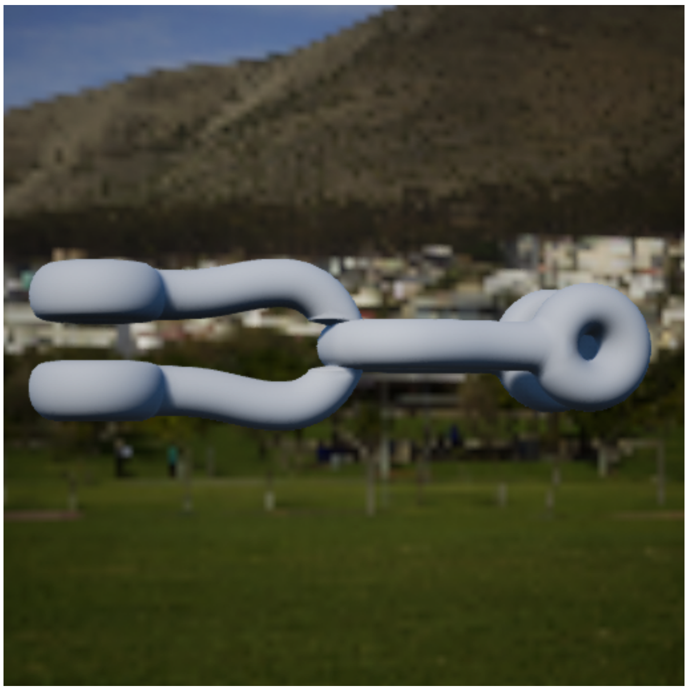


Figure 3.1: A sample synthetic image rendered on top of HDRI background

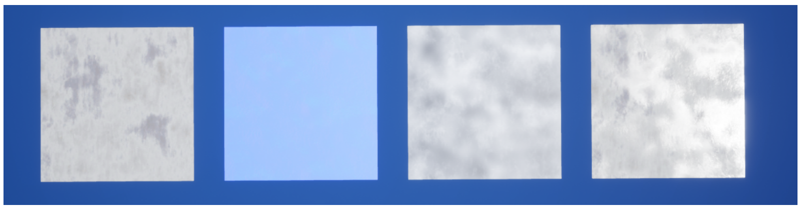


Figure 3.2: Four styles of texture utilized in worn chain shackles case study

- NOTX_WBG: the background variation is enabled. However, there is no texture.
- WTX_WBG: both background and texture are on.
- WBG_WTX_GM: random background, texture, geometric. The most complicated dataset.

Apply Deep Learning on synthetic data

eSmart utilized CSRN (Chain Shackle Regression Net) as their own solution architecture. The CSRN is a self-defined CNN architecture inherited from LS-Net (Line Segmentation Net) (Nguyen, Jenssen, & Roverso, 2020). To build a vanilla regression model, the CSRN replaced the output layer of the LS-Net with a linear perceptron, while reserved 16 convolution-layer architecture of the LS-Net. CSRN consisted of 16 layers, equally distributed into four blocks. Each block used ReLU as its activation function. The CSRN's details is listed below

- Block 1: 64 filters each layer, each filter has the size of 3*3
- Block 2: 128 filters each layer, each filter has the size of 3*3

- Block 3: 256 filters each layer, each filter has the size of 3*3
- Block 4: 512 filters each layer, each filter has the size of 3*3

Each model was trained from scratch, without inheriting any pre-trained weights. Besides, a margin μ was introduced in the mean absolute error loss function to compromise the loss. In other words, this solution accepted a margin between the prediction and the truth. For example, with the margin of 5%, if the ground truth is 80% worn and the prediction lies between [75%, 85%], it accepts that prediction and yields a zero loss. Put differently, the smaller the margin, the stricter the model. In this case study, the mean absolute error L is offseted with the loose margin μ of 0.2.

$$L = \frac{1}{B}\sum_{i=1}^{B}\max(|\hat{y}_i - y_i| - \mu, 0) \tag{3.1}$$

Mean absolute error belongs to L1 loss class. L1 losses are used to minimize the error which is the sum of the all the absolute differences between the true value and the predicted value. They penalize fairly both for the small residual and big residual compare to the labeling.

eSmart assumed that a model trained on a dataset with simple 3D variation settings can outperform those trained on datasets with more complicated settings. In other words, as Occam's razor has mentioned, if two solutions offer the same output, the most simple solution should be chosen. In order to test this, three first datasets were chosen for cross training and testing.

Results and early analysis

Early analysis, models trained on datasets with simple settings performed poorly on those having complicated settings. The early results are shown in figure 3.3, where complexity of dataset settings are ordered from left to right, top to bottom. Therefore, as the first analysis, the assumption of "less is more" hardly work for worn chain shackles case study.

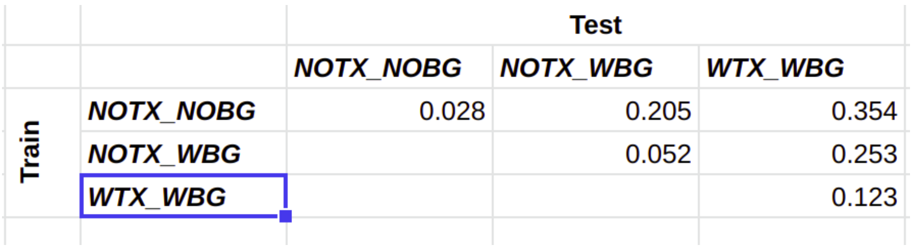
Hence, another model was trained on the most complicated dataset, WBG_WTX_GM, with the same settings defined above. The loss is depicted as the figure 3.4

The trained model later was tested on few real world images. However, the predictions were suspicious and unreliable. For example, in figure 3.5, the model predicted 80% worn percentage although the chain shackles looked normal.

NMLab applied Canny edge detector between images to conduct an early analysis conducted by Nordic Media Lab (Marius, 2019) concluded that synthetic data is far better than the real-world data. In other words, the synthetic images are too simple for the model to learn compared to real-world data. For example, the lines in synthetic data are sharper and straighter than the real-world data as in figure 3.6.

3.5 Summary

Many research has figured to change the variations of 3D factors as the effort of closing the the synthetic gap. However, usually, those number of combinations is too big, hence, can not be covered completely in the experiment. The common solution is conduct try-and-fail experiments or arbitrarily pick some combinations to report like the work of (Lundmark, 2019). To the best of our knowledge, there is no literature reporting the application of network visualizations, synthetic data on preventive maintenance. Therefore, utilizing network visualization



		Test		
		NOTX_NOBG	***NOTX_WBG***	***WTX_WBG***
Train	***NOTX_NOBG***	0.028	0.205	0.354
	NOTX_WBG		0.052	0.253
	WTX_WBG			0.123

Figure 3.3: Loss results of models trained and tested on three datasets

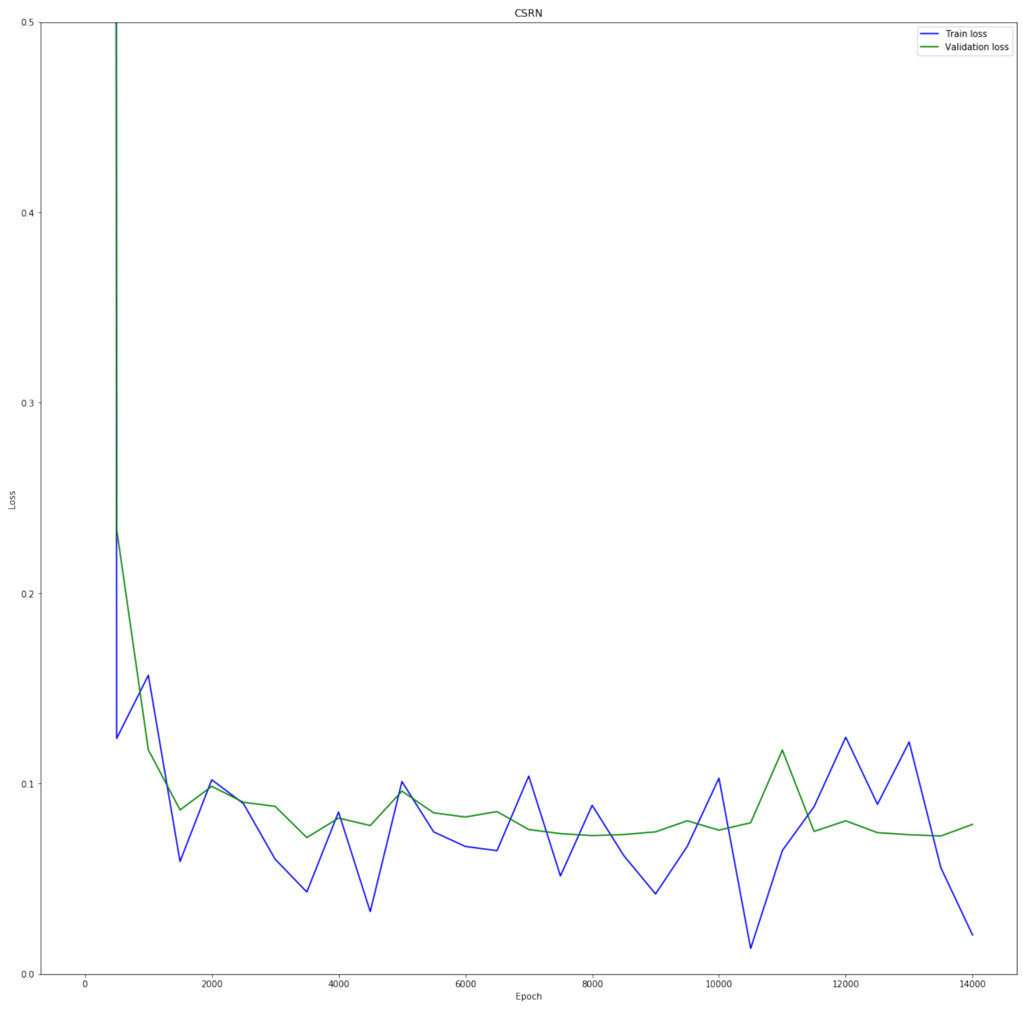


Figure 3.4: Train loss and validation loss of CSRN

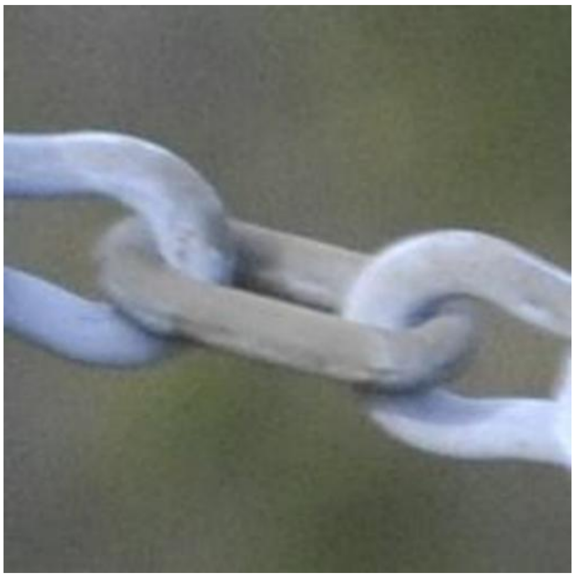


Figure 3.5: Image of normal chain shackles which CSRN predicts 80% worn

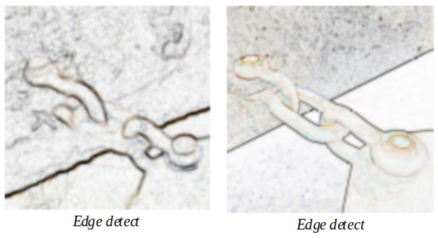


Figure 3.6: Outputs of Canny edge detector on real-image (left) and synthetic image (right)

could provide some insights about how a network trained on synthetic data behaves on synthetic data and real-world data.

Furthermore, as (Lathuilière et al., 2019) stated in their vanilla regression study, network architectures and adjustments of network variants can improve the performance of the output model. Therefore, we can consult this research as our foundation for the case study of "worn chain shackles" at eSmart.

We argue that the application of the CSRN might be insufficient to learn critical patterns of the synthetic dataset. Because the network's capacity hardly can cover the complexity of worn chain shackles domain. Besides, offseting loss function with margin μ of 0.2, which accounts nearly 1/5 of the expected output range of (0-95%), can discredit the trained model on unseen data.

Chapter 4

Methodology

After the consulting the literature in related work and gaining the overview of both pros and cons from previous work, this chapter intends to propose a our new approach. Firstly, we state our objectives, regarding both scientific gain and industrial benefit. We briefly provide readers with our experimental setup and development environment.

4.1 Objectives

According to the related work and the initial approach at eSmart, we would like to achieve those goals in our experiment:

- Select a network architecture that learns at best the pattern of "worn chain shackles" case study at eSmart.
- Produce a practical, effective models that can predict the wear joints on synthetic images without offsetting the loss.
- Provide Nordic Media Lab with analysis regarding the learned patterns of trained models in order to avoid time-consuming and arbitrary choices of 3D variations.
- Study the impacts of deep learning and transfer learning on this specific regression problem.

4.2 Development environment and worn chain shackles problem definition.

4.2.1 Input and output

Input: dataset of synthetic images

Output: a regression model that can predict how worn the joint of the chain shackles (in percentage, ranging from 1% to 95%)

4.2.2 Environments

Hardware

GPU: NVIDIA GTX 1080 Ti, 11 GB memory, 3584 CUDA cores

Software

- Tensorflow 2 with Keras syntax
- Jupyter Lab
- Python 3.7

4.3 Datasets

4.3.1 Dataset 1 - Dataset WBG_WTX_WGM

Settings: with background, with texture, with geometric as mentioned as section 3.4
Image size: 512*512 RGB
Number of image: 20,000
Training split: 60/20/20 (Training set/Validation set/Test set)

4.3.2 Dataset 2

Settings: adjustment of dataset 1 settings based on analysis between eSmart and Nordic Media Lab.
Image size: 512*512 RGB
Number of image: 30,000
Training split: 60/20/20 (Training set/Validation set/Test set)

4.4 Experimental setup

4.4.1 Experiment 1: Network architecture selection

We argue that the inheritance of simple CSRN might be insufficient to learn critical patterns of worn chain shackle. Hence, we need to select another architectures. We choose VGG16 and ResNet50 as (Lathuilière et al., 2019) suggested in their comprehensive study of deep regression. Besides, because eSmart Systems has an ambition of autonomous AI-driven drone inspectors, it favors an deep network architecture with powerful capacity but fast computation and light weight that can be deployed on mobile and/or micro-computers. Therefore, we enlist MobileNet as our candidate.

Base networks, with mainly the feature extractors, are achieved from Tensorflow applications package. The regression layer contains an layer of global average pooling (GAP) followed by a vanilla output, a.k.a a perceptron with linear activation (Lathuilière et al., 2019). Furthermore, as (Hinterstoisser et al., 2018) reported that feature extractors pre-trained weights are sufficient enough without further fine-tuning, we would like to test his research finding in our experiment. The details are listed in table 4.1.

At the end of experiment 1, two architectures which have lowest test losses will be chosen for network visualization in experiment 2.

Base network	Source	Fine-tuning	Regression Layer
MobileNetv2	Tensorflow Hub	Yes	GAP—Vanilla
ResNet50	Tensorflow Hub	Yes	GAP—Vanilla
VGG16	Tensorflow Hub	Yes	GAP—Vanilla
MobileNetv2	Tensorflow Hub	No	GAP—Vanilla

Table 4.1: Experiment 1: Network architecture descriptions

4.4.2 Experiment 2: Feature maps visualizations and 3D variations adjustments

Although Nordic Media Lab suggested the idea of applying Canny line detector filter to discover difference between synthetic data and real-world data, it was a single, subjective selection of a line detector. On the other hand, a deep learning models trained on synthetic data contains kernels with more meaningful and tuned weights. As mentioned in 3, CNNs learning experience and behaviors can be exposed via the first few layers. We can visualize the feature maps of first few layers of the network to understand the learned patterns. This systematic approach is more problem-oriented as it is designated to gain insights of what the trained networks are looking at.

We prepare beforehand five pairs real-world images and their synthetic version. Those synthetic samples are generated with dataset 1's settings mentioned above.

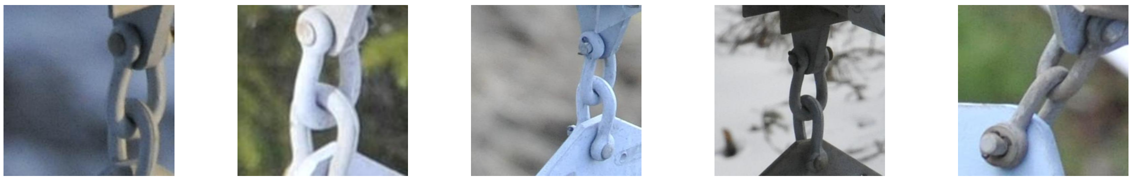


Figure 4.1: Five real-world images: Sample 1a, 2a, 3a, 4a, 5a

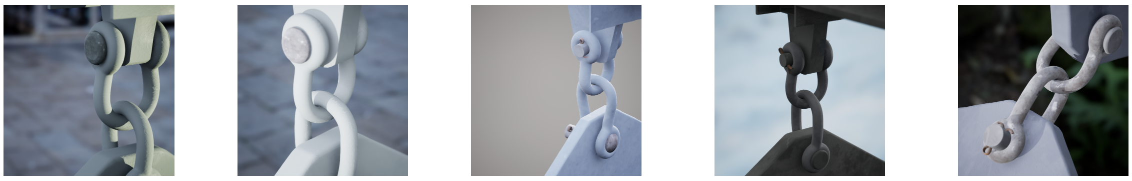


Figure 4.2: Five synthetic images generated with dataset 1's settings: Sample 1b, 2b, 3b, 4b, 5b

For each pair, we forward each pair through a trained model and collect the feature maps of its very first layers. Below are the layer names that we choose to visualize in experiment 2 (their names as in tensorflow.keras.applications network summary):

- VGG16: block1_conv1, block2_conv1, block3_conv1, block4_conv1, block5_conv1
- MobileNet layers: Conv1, block_1_project, block_2_project, block_3_project, block_4_project, block_5_project.
- ResNet50: conv1_conv, conv2_block3_3_conv, conv3_block4_3_conv, conv5_block5_3_conv, conv5_block5_5_conv

We will consult with Nordic Media Lab for their expertise on the output. Later, we generate dataset 2, a new dataset with 3D variation adjustments based on the analysis we harvest from the analysis.

4.4.3 Experiment 3: Further training on new dataset selection

Besides two architectures selected from experiment 1, we would like to experiment on self-constructed ResNet50 architecture based on built-in that weights initialized randomly.

Base network	**Source**	**Fine-tuning**	**Regression Layer**
MobileNetv2	Tensorflow Hub	Yes	GAP—Vanilla
ResNet50	Tensorflow Hub	Yes	GAP—Vanilla
ResNet50	Camisential Github	Yes	GAP—Vanilla

Table 4.2: Experiment 3: Network architecture descriptions

4.5 Training specifications

Max number of training: 1000 epochs.

Each epoch run on training set with s steps, where step s is defined as:

$$s = floor(\frac{Dataset\_size}{Batch\_size}) \tag{4.1}$$

Besides, in order to prevent overfitting, early stopping is introduced. The early stopping is based on validation loss, with a patience P. It means after P epochs since the lowest validation loss is found, if there is no solution which has lower validation loss, the training will be halted and the best loss solution will be achieved.

Only In this experiment, in order to compare performances of trained models at best, we use only mean absolute error. In other words, the margin μ as depicted in 3.4 is removed. Loss Mean Absolute Error.

As data augmentation does not affect the labels of images, we apply data augmentation on the data with the combination of rotation (range 45 degree), width cropping (range 0.15) and width cropping (range 0.15). The hyper-parameter settings are selected from Tensorflow Tutorials page.

To apply transfer learning, we inherited the pre-trained weights on ImageNet. Those weights can be acquired from Tensorflow.keras.application.

Chapter 5

Results

5.1 Experiment 1

Network architecture	Train loss	Validation loss	Test loss
ResNet50	0.0065	0.0077	0.0078
MobileNet	0.0082	0.0087	0.0085
VGG16	0.2272	0.2296	0.2247
MobileNet (No fine-tuning)	0.1971	0.1976	0.2298
CSRN	0.0431	0.0717	0.0717

Table 5.1: Experiment 1: Train loss, validation loss and test loss reported from different network architectures

5.2 Experiment 2

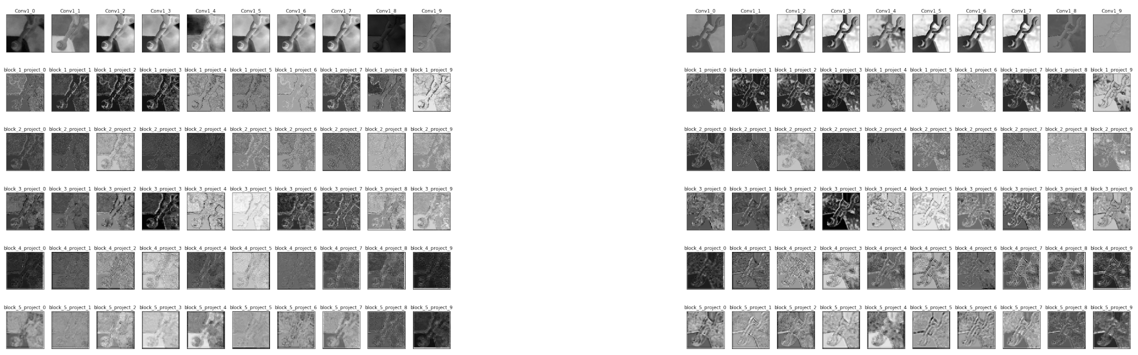


Figure 5.1: Experiment 2: MobileNet's feature maps visualization of sample 5a and 5b

On the forward pass activation outputs of trained ResNet50 and MobileNet, the networks pay attention to both chain shackles and background noise at the few first convolution layers. Later, the deeper the information pass, the more attention the network pay to the boundaries of the chain shackles. The forward pass of the synthetic image shows an overall distribution of lighting on the surface of the chain shackles, which differs from the partly illuminated condition on the real-world image.

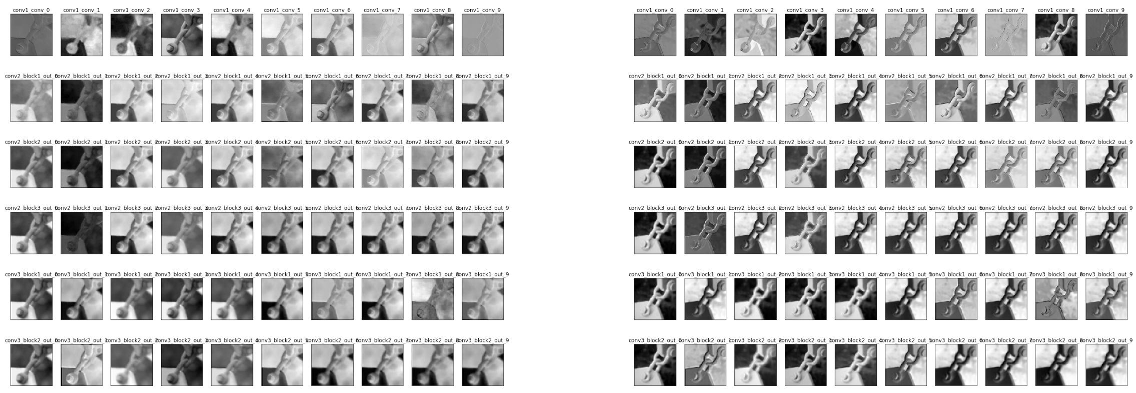


Figure 5.2: Experiment 2: ResNet50's feature maps visualization of sample 5a and 5b

However, on the real images, the background noises are reduced faster during the forward pass, which allows the network to detect the primary target objects faster. It can be explained because the camera focuses more on the chain shackles, which makes the background blurry. On the contrary, on the synthetic images, the camera focuses on both backgrounds and chain shackles. Consequently, the background noise is hardly eliminated during the forward pass. Therefore, the camera focus should be adjusted in the game engine to generate better synthetic images.

Please refer to appendix A and appendix B for other feature map visualization.

5.3 Experiment 3

Network architecture	**Train loss**	**Validation loss**	**Test loss**
ResNet50 (self-constructed)	0.0104	0.0104	0.0101
ResNet50	0.0095	0.0098	0.0095
MobileNet	0.0077	0.0086	0.0086

Table 5.2: Experiment 3: Train loss, validation loss and test loss reported from different network architectures

5.4 Training curves

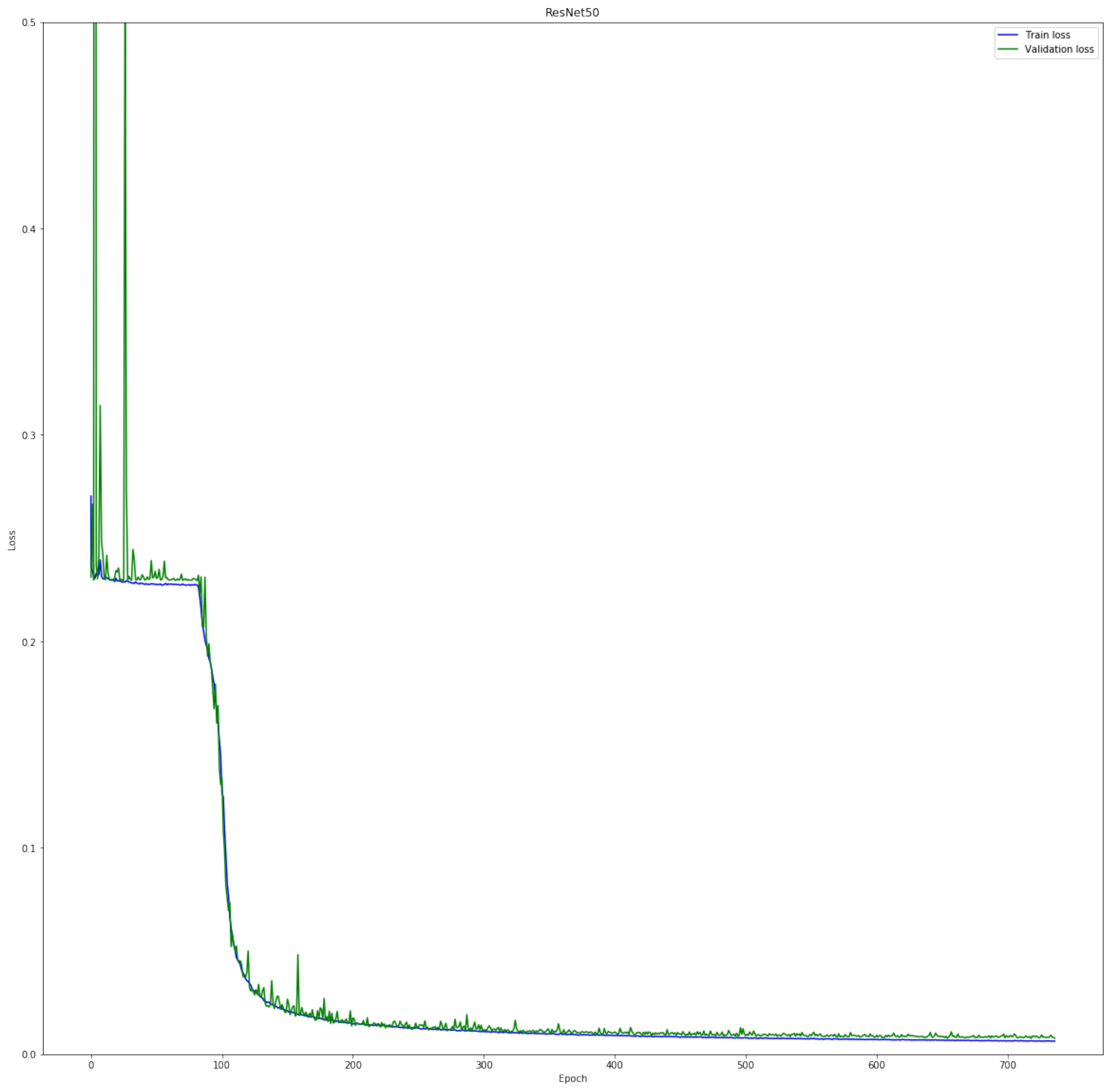


Figure 5.3: Experiment 1: Plot of train and validation loss of ResNet50

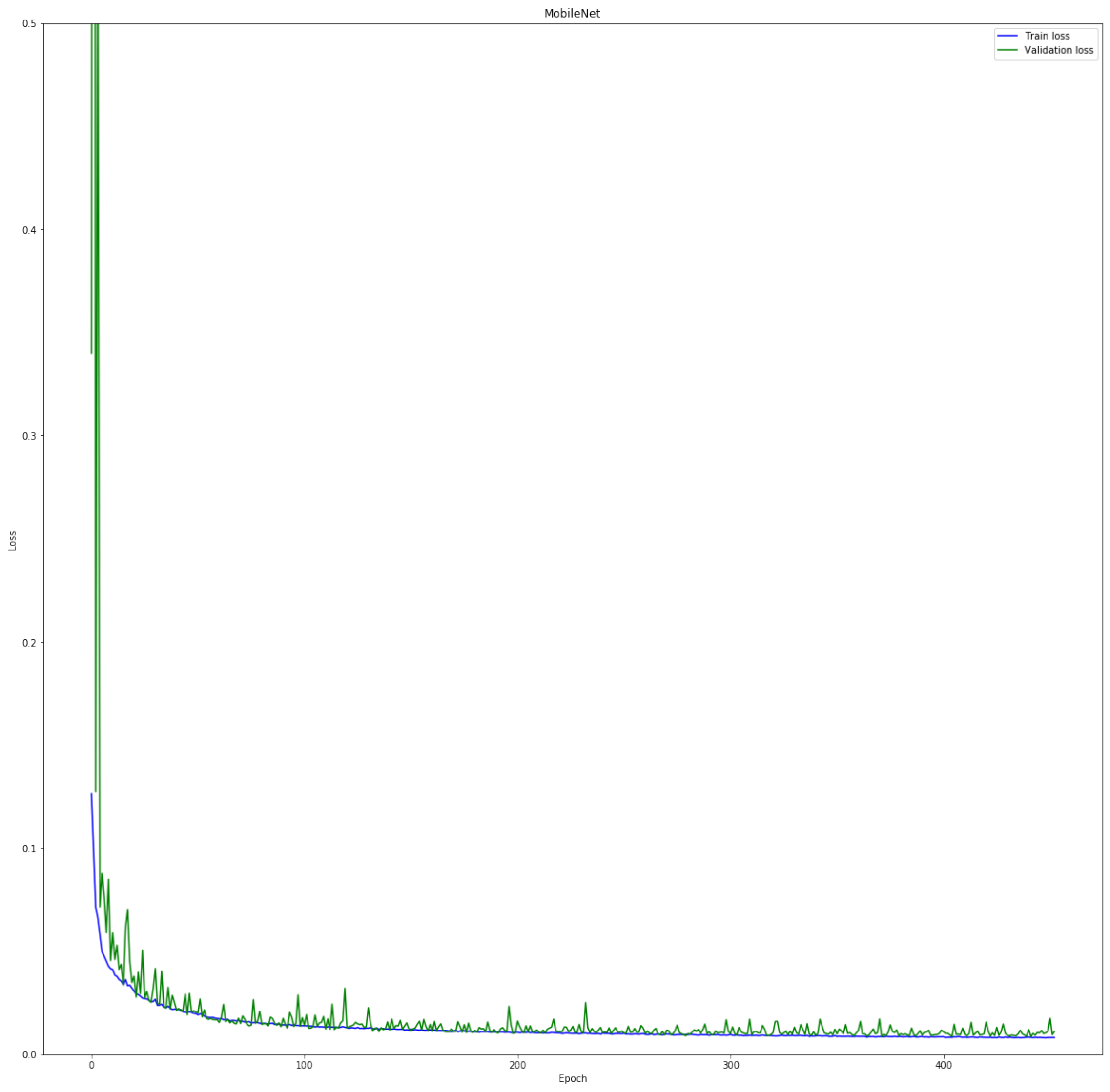


Figure 5.4: Experiment 1: Plot of train and validation loss of MobileNet

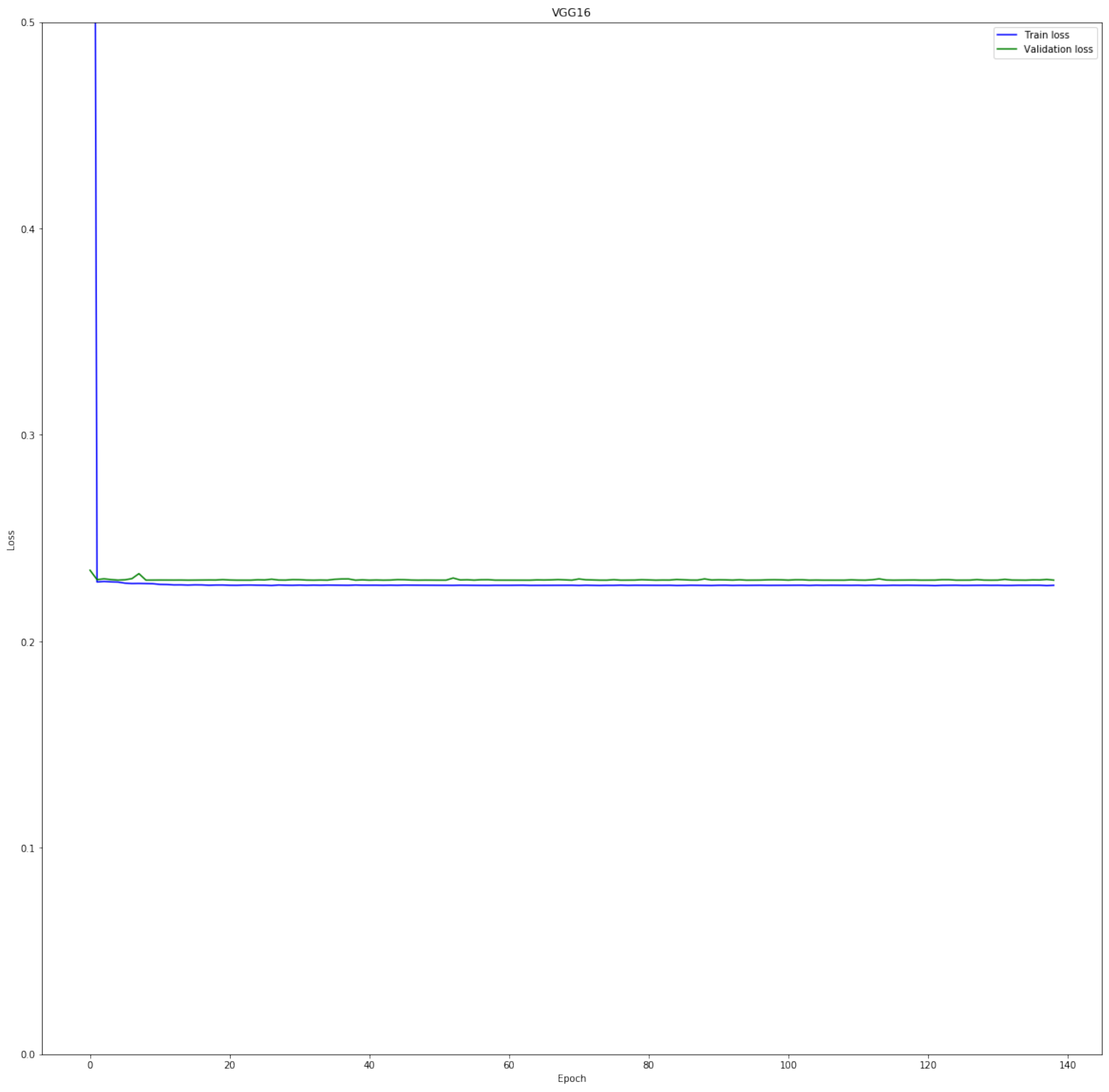


Figure 5.5: Experiment 1: Plot of train and validation loss of VGG16

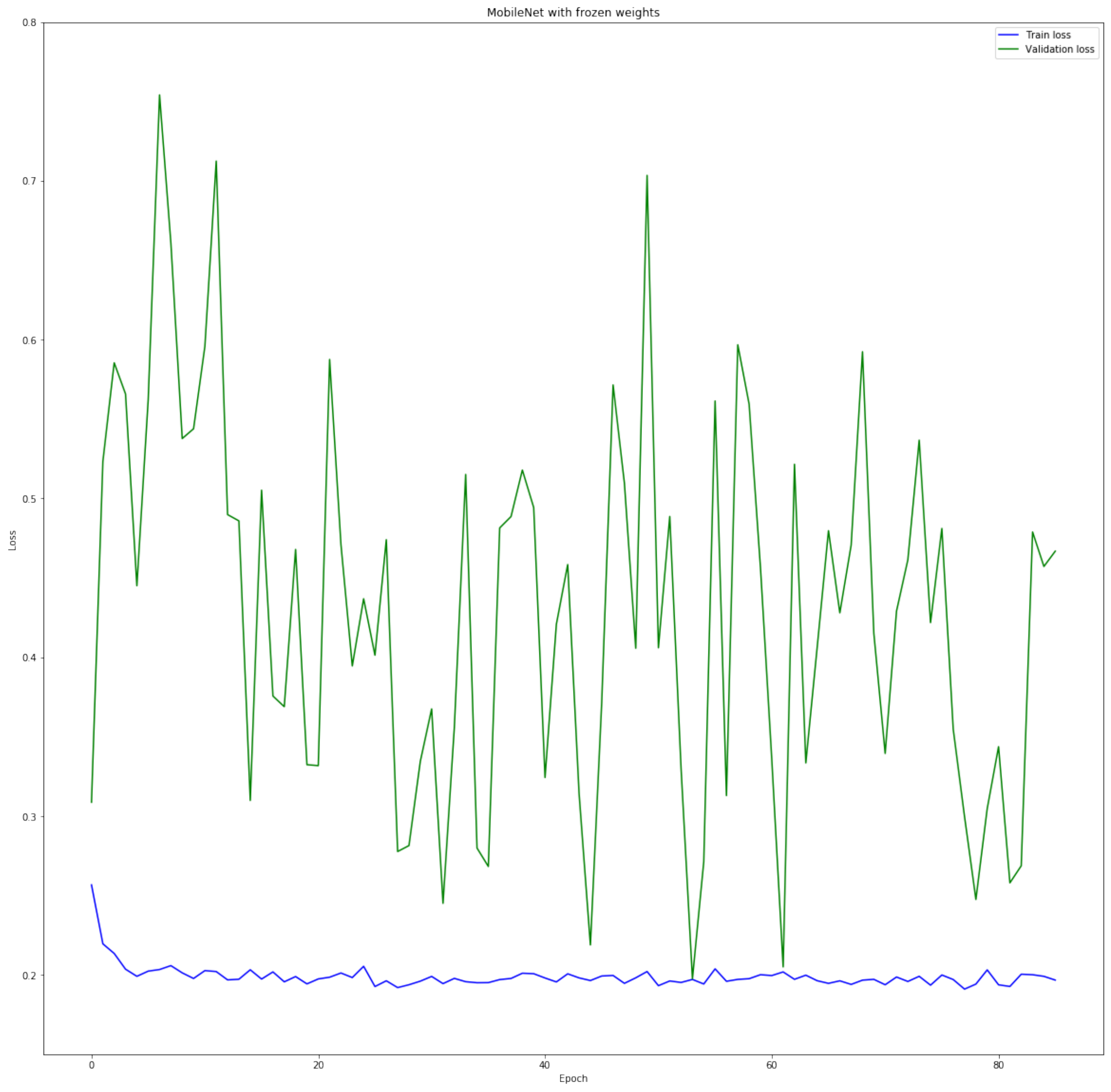


Figure 5.6: Experiment 1: Plot of train and validation loss of MobileNet without fine-tuning

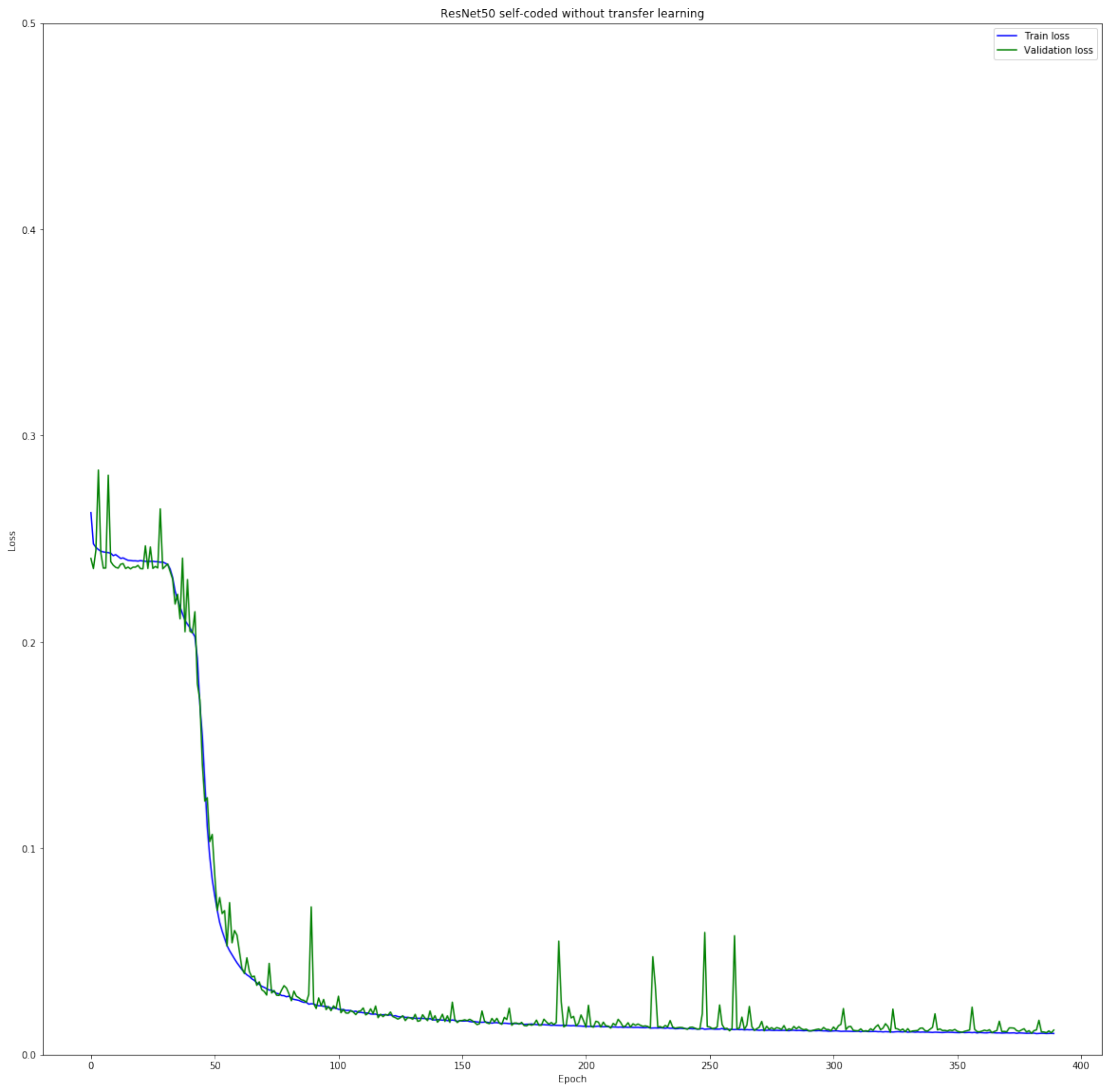


Figure 5.7: Experiment 3: Plot of train and validation loss of self-constructed ResNet50 without transfer learning

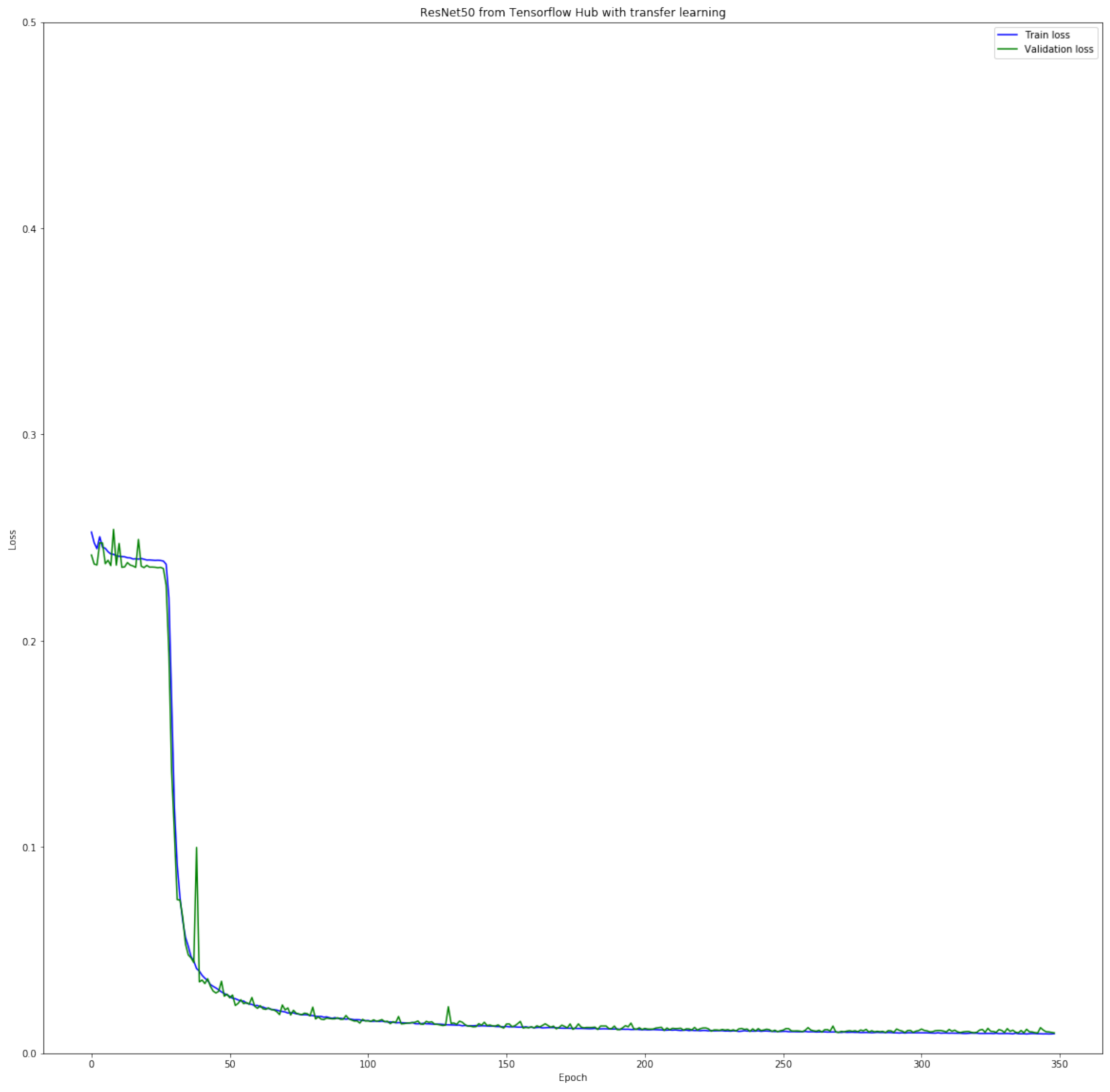


Figure 5.8: Experiment 3: Plot of train and validation of ResNet50 with transfer learning

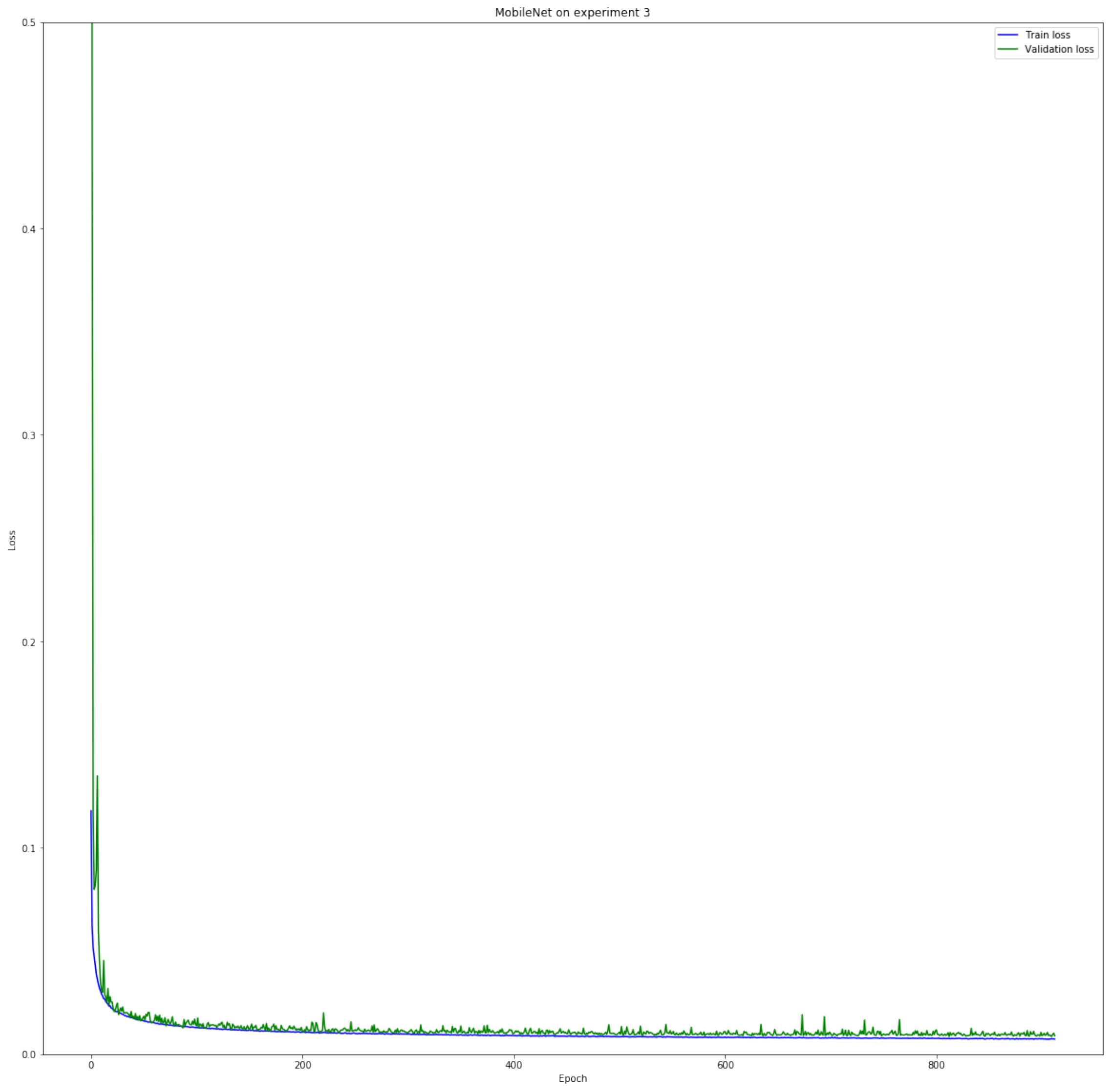


Figure 5.9: Experiment 3: Plot of train and validation loss of MobileNet with transfer learning

Chapter 6

Discussion

6.1 Experiment 1

On the original dataset 1, the ResNet50 has the lowest loss on both validation set and test set, which is 0.0077 and 0.0078 successively. The fine-tuned MobileNet comes second with a higher loss of 7e-4 compared to the ResNet50. Besides this slightly less competitive performance, the fine-tuned MobileNet takes approximately 15 epochs to converge to the feasible solution compared to 120 epochs of the ResNet50. Additionally, during the training, the fine-tuned MobileNet takes only 441 epochs and 97 hours to reach its lowest loss on the validation set. On the other hand, the ResNet50 needs 718 epochs and 162 hours to accomplish its lowest loss on the validation set. Regarding the training time per epoch, the fine-tuned MobileNet is faster than the ResNet50 by 18 seconds.

Both MobileNet's model and ResNet50's model show similar losses during training testing phases, which indicates that these two architectures hardly prone to overfit on the training set. During the training phase, MobileNet's validation loss fluctuates in a small range, while the curve of ResNet's validation loss is more rigid, sometimes, over 0.5. We can conclude that MobileNet is less likely to be overfitted during the training phase. Thirdly, although transfer learning with ResNet50 and MobileNet show promising outcomes on worn chain shackles dataset, their validation losses fluctuate noticeably at the very first epochs. Deep neural networks need their time to adapt to the complexity of this worn chain shackles task.

The MobileNet without fine-tuning weights shows the highest loss of 0.2298. Furthermore, as depicted in 1.2, this training method tends to cause severe overfitting depicted by turbulent validation loss values. Therefore, although transfer learning benefits of the inheritance of weights pre-trained on millions of images, they need to be fine-tuned during training in order to solve a worn chain shackles deep learning task. The idea of leaving pre-trained weights unchanged during training from (Hinterstoisser et al., 2018) hardly works on our case study.

The VGG16 architecture experiences from underfitting on Experiment 1. Besides, as the number of epochs increases, VGG16 training loss and validation loss shows little or no improvements. It could be explained by the insufficient capacity of the VGG16 caused by the narrow depth of its architecture. Hence,the VGG16 is unable to produce plausible mathematic representations for the complexity of the dataset.

From the learning curves of all network architectures except fine-tuned MobileNet, we can see a plateau of 0.22. Fortunately, deep neural network architectures like ResNet50 can get past this plain elevation of the search solution and reach a better local optimum, which is 0.0077. However, we need to provide the ResNet50 with a more significant patience threshold,

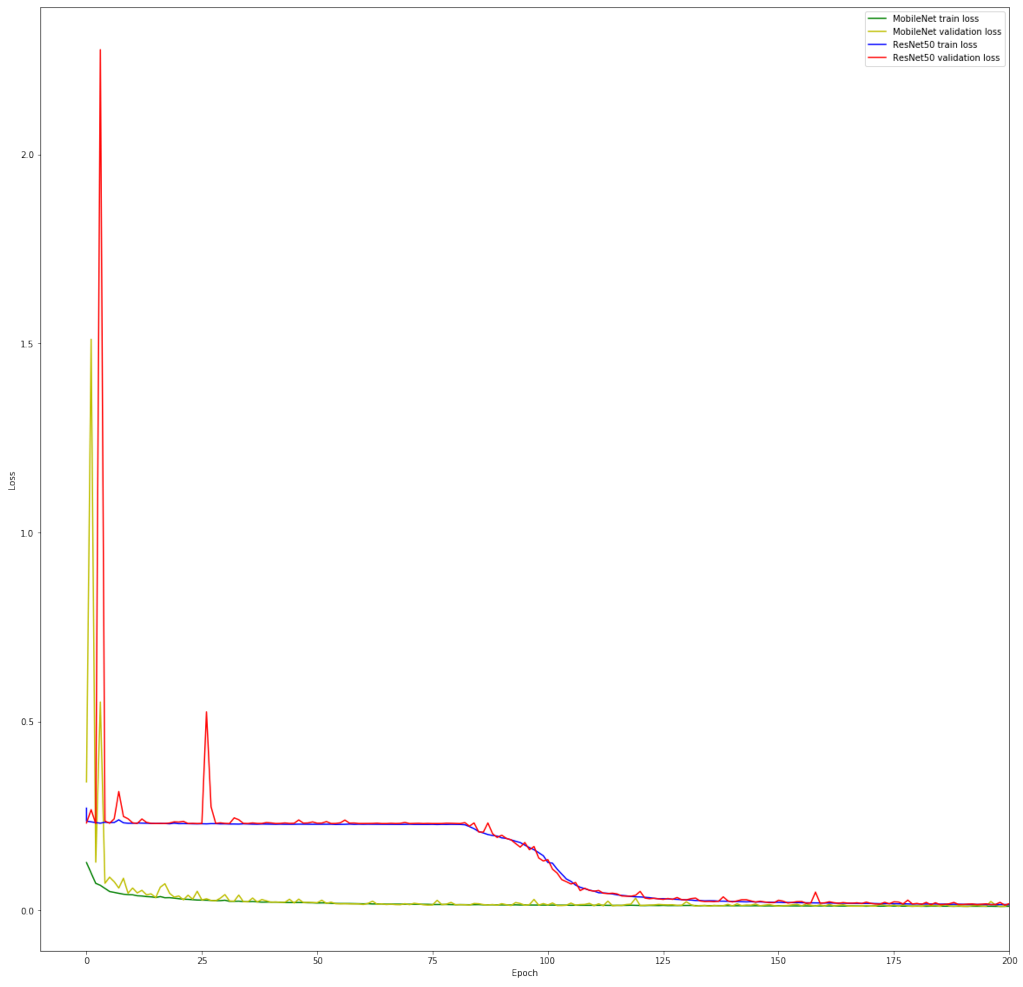


Figure 6.1: Experiment 1: Comparision between ResNet50 and MobileNet on the first 180 epochs

in this case, 100. Put differently, when early stopping is applied on very deep neural networks like ResNet50, it is advisable to set a bigger stopping threshold for the "worn chain shackles" case study.

6.2 Experiment 2

First and foremost, both ResNet50 and MobileNet models are learning the correct patterns from synthetic images. For example, the deeper the layer, the more blurry of background information and the sharper the shackles' boundaries. It indicates that trained kernels tend to eliminate background noises. Moreover, at some feature maps, we can see a bolder output of the joints. Hence, it proves that the trained networks correctly identify and prioritize pixels of joints where the worn faults occur.

Because of the high resolution of the image generator, synthetic chain shackle objects have sharper boundaries than those shackles in the real-world domain. This indicates a synthetic gap caused by the high resolution of the synthetic images. Along the forward pass, the information loss of the synthetic images is less than the information loss of the real-world images. Hence, our systematic visualization agrees with the assumption of Nordic Media Lab, saying that the synthetic data is simpler for the network to learn regarding the resolution.

Secondly, our visualization reveals another synthetic gap caused by the camera focus on the background and the chain shackles objects. The models trained on synthetic image suffer from background noise, which is hardly eliminated along the feature maps. Because the camera focuses on the main object of interests and the background equally. While in the real-world domain, the background's focus is more blurry, which helps the trained model to eliminate background noise faster.

6.3 Experiment 3

Within transfer learning models, the MobileNet has the lowest losses on both test set and validation set. The ResNet50 stands second in this experiment, with 12x10-4 points higher on validation loss, and 9e-4 points higher on test loss than those of the MobileNet.

The MobileNet remains the same fast pace of convergence to feasible local optimum as in experiment 1. However, in this experiment, it takes 812 epochs and approximately 13.5 days to find its most optimal solution of 0.0086.

The ResNet50 takes 342 epochs, approximately 211 hours, 37 minutes each epoch to reach the optimal solution.

The self-reconstructed ResNet50 architecture without transfer learning has more unsteady validation loss. It reaches its feasible optimum of 0.0104, after 380 epochs and approximately 236 hours. It is noticeable that its loss is slightly higher than that of the the ResNet50 acquired from Tensorflow Hub, by just 6x10-4. This pattern indicates that without transfer learning, ResNet50 architecture still can reach a feasible optimum, despite a higher possibility of overfitting on the training set.

There is evidence showing that adjusting 3D variations in experiment 2 reduces the complexity of the learning problem. Firstly, although there is still a plateau of 0.24 that both two ResNet architectures need to overcome, they start to escape the plateau after 45 epochs rather than 120 epochs, as in experiment 1. Secondly, the ResNet50 with transfer learning starts to converge faster to its best solution, after running for 30 epochs, rather than 120 epochs, as on the dataset 1.

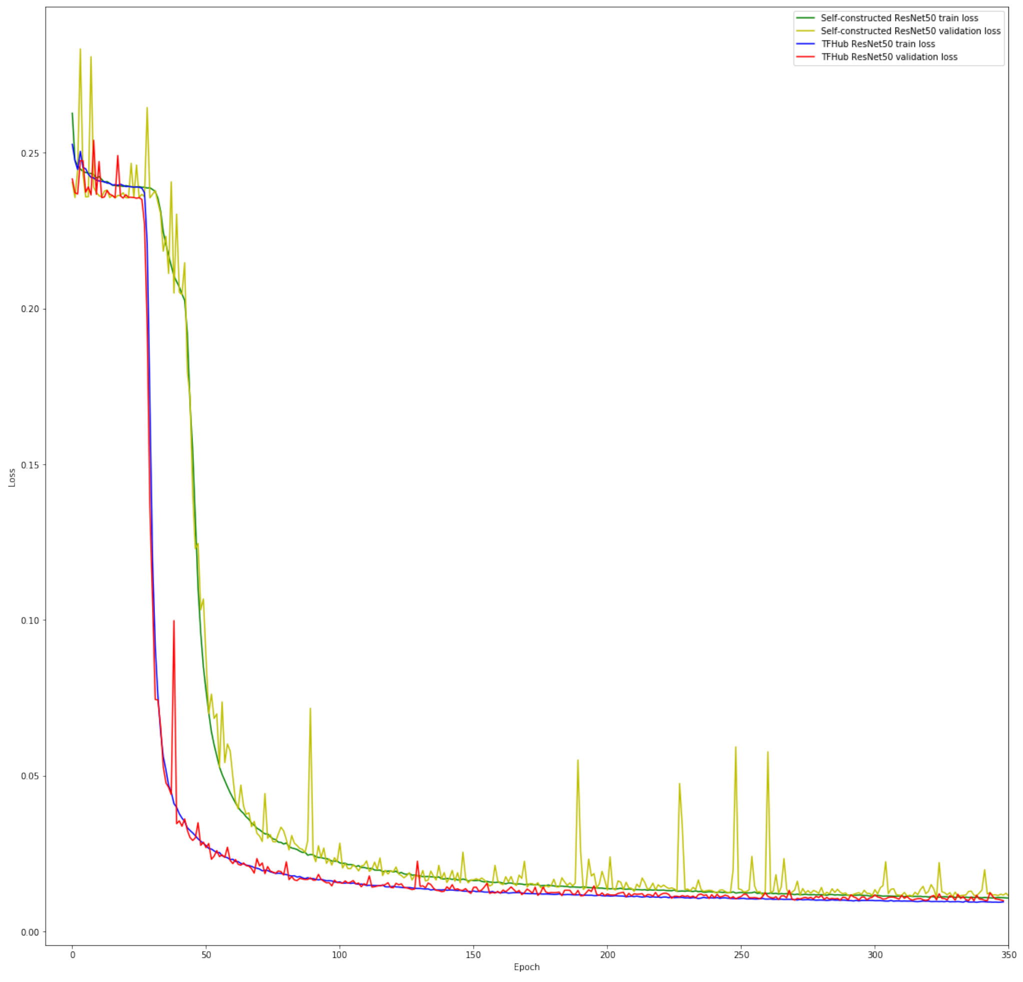


Figure 6.2: Experiment 3: Comparison between ResNet50 achieved from Tensorflow Hub with transfer learning and self-constructed ResNet50 without transfer learning on the first 350 epochs

Chapter 7

Conclusion and future work

In this chapter, we conclude our work and answer the research question as we stated in section 1.2, chapter 1. We also suggest directions for further development of this thesis.

7.1 Conclusion

RQ How to bring deep learning and synthetic data into practice to improve a power line inspection task?

In this thesis, by introducing transfer learning, we have successfully used deep learning to maximize the practice of synthetic data on worn chain shackles regression, a case study of power line inspection at eSmart Systems AS. We reduce the current solution loss by six percentage points, from 0.0717 to 0.0077, without using a margin μ to offset the mean absolute error loss function. Therefore, the trained models can predict more precisely on synthetic data.

SQ1 What are deep learning architectures that can utilize synthetic data at best?

The ResNet50 and the MobileNet architectures are potential deep neural networks that can effectively address the worn chain shackles case study. On the contrary, narrow neural networks like the VGG16 are lack of capacity to solve the complexity of worn chain shackles case study.

SQ2 How do the insights of deep neural networks to can reduce the negative effect of *reality gap*?

We have detected and reduced the negative impact of so-called synthetic gap, the difference between the real-world domain and the synthetic domain, by using analysis based on network visualizations. We use the insightful analysis of two network architectures: the ResNet50 and the MobileNet to adjust 3D variations for synthetic data. The analysis shows that backgrounds with a clear focus in synthetic images can add unnecessary noise and complicate the complexity of worn chain shackles problem. Our approach avoids picking 3D variations arbitrarily and conducting time-consuming searches for all the 3D variable combinations. To the best of our knowledge, this approach has not been introduced in the literature before.

SQ3 Which are the rule-of-thumb deep learning practices that can maximize the benefit of synthetic data?

First and foremost, fine-tuning pre-trained weights successfully exploits the potential of the deep neural networks. Although the feature extractors are trained on benchmarking datasets like ImageNet, they still need to adapt to new deep learning tasks. Otherwise, the models are prone to suffer from severe overfitting. Moreover, for deep neural networks like the ResNet50, it is advisable to define a higher early stopping threshold that can help the model get past the plateau and reach the feasible optimum.

7.2 Future Work

Regarding academic contributions, there are several directions along that we can further discover in later research. Firstly, it is promising to use pre-trained weights of other benchmarking datasets, for example, COCO. Secondly, it is rewarding to test the impact different network optimizers on the training's convergence speed as (Lathuilière et al., 2019) suggested. Thirdly, fine-tuning the other network hyper-parameters such as batch size, learning rate can effectively reduce the loss of the deep learning architectures.

Regarding industrial interests, it is necessary to make a systematic test on real-world images. In detail, we should cooperate with inspectors from energy provider companies to formulate a standard definition of worn chain shackles. Later, a repository of worn chain shackles images should be collected and labeled by a committee of experienced experts.

References

Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York: Springer.

Borrego, J., Dehban, A., Figueiredo, R., Moreno, P., Bernardino, A., & Santos-Victor, J. (2018, July). Applying Domain Randomization to Synthetic Data for Object Category Detection. *arXiv:1807.09834 [cs, stat]*. Retrieved 2019-08-01, from http://arxiv.org/abs/1807.09834 (arXiv: 1807.09834)

Derbeken, J. V. (2019, November). Long-Term Wear Found on PG&E Line That Sparked Camp Fire. *NBC Bay Area*. Retrieved 2020-06-14, from https://www.nbcbayarea.com/news/local/long-term-wear-found-on-pge-line-that-sparked-camp-fire/2151906/

Dwibedi, D., Misra, I., & Hebert, M. (2017, August). Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection. In *arXiv:1708.01642 [cs]*. Retrieved 2019-08-23, from http://arxiv.org/abs/1708.01642 (arXiv: 1708.01642)

Georgakis, G., Mousavian, A., Berg, A. C., & Kosecka, J. (2017, February). Synthesizing Training Data for Object Detection in Indoor Scenes. In *arXiv:1702.07836 [cs]*. Retrieved 2019-08-26, from http://arxiv.org/abs/1702.07836 (arXiv: 1702.07836)

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Cambridge, Massachusetts: The MIT Press.

He, K., Zhang, X., Ren, S., & Sun, J. (2016, June). Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 770–778). doi: 10.1109/CVPR.2016.90

Hinds, M. D. (1985, February). Preventive Maintenance: A Checklist. *The New York Times*. Retrieved 2020-06-14, from https://www.nytimes.com/1985/02/17/realestate/preventive-maintenance-a-checklist.html

Hinterstoisser, S., Lepetit, V., Wohlhart, P., & Konolige, K. (2018). On Pre-Trained Image Features and Synthetic Images for Deep Learning. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 0–0). Retrieved 2019-08-23, from http://openaccess.thecvf.com/content_eccv_2018_workshops/w6/html/Hinterstoisser_On_Pre-Trained_Image_Features_and_Synthetic_Images_for_Deep_Learning_ECCVW_2018_paper.html

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... Adam, H. (2017, April). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*. Retrieved 2020-06-14, from http://arxiv.org/abs/1704.04861 (arXiv: 1704.04861)

Kingma, D. P., & Ba, J. (2017, January). Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*. Retrieved 2020-06-15, from http://arxiv.org/abs/1412.6980 (arXiv: 1412.6980)

Klosterman, C. (2016, April). Vision helps perform predictive maintenance. *Vision Systems Design*. Retrieved 2020-06-14, from https://www.vision-systems.com/home/article/16736828/vision-helps-perform-predictive-maintenance

Lathuilière, S., Mesejo, P., Alameda-Pineda, X., & Horaud, R. (2019, February). A Comprehensive Analysis of Deep Regression. *arXiv:1803.08450 [cs]*. Retrieved 2020-02-24, from http://arxiv.org/abs/1803.08450 (arXiv: 1803.08450)

Lee, W. J., Wu, H., Yun, H., Kim, H., Jun, M. B. G., & Sutherland, J. W. (2019, January). Predictive Maintenance of Machine Tool Systems Using Artificial Intelligence Techniques Applied to Machine Condition Data. *Procedia CIRP*, *80*, 506–511. Retrieved 2019-11-20, from http://www.sciencedirect.com/science/article/pii/S2212827118312988 doi: 10.1016/j.procir.2018.12.019

Lu, T., Huyen, A., Nguyen, L., Osborne, J., Eldin, S., & Yun, K. (2019, May). Optimized training of deep neural network for image analysis using synthetic objects and augmented reality. In *Pattern Recognition and Tracking XXX* (Vol. 10995, p. 109950I). International Society for Optics and Photonics. Retrieved 2019-08-07, from https://www.spiedigitallibrary.org/conference-proceedings-of-spie/10995/109950I/Optimized-training-of-deep-neural-network-for-image-analysis-using/10.1117/12.2522198.short doi: 10.1117/12.2522198

Lundmark, L. (2019). *Synthetic Meta-Learning Learning to learn real-world tasks with synthetic data* (Doctoral dissertation, KTH Royal Institute of Technology). Retrieved 2019-08-07, from http://www.nada.kth.se/~ann/exjobb/lukas_lundmark.pdf

Marius, E. (2019, June). *Synthetic data for Deep Learning models at eSmart Systems AS* (Tech. Rep.). Halden: Smart Innovation Norway.

Mayer, N., Ilg, E., Fischer, P., Hazirbas, C., Cremers, D., Dosovitskiy, A., & Brox, T. (2018, September). What Makes Good Synthetic Training Data for Learning Disparity and Optical Flow Estimation? *International Journal of Computer Vision*, *126*(9), 942–960. Retrieved 2019-08-01, from http://arxiv.org/abs/1801.06397 (arXiv: 1801.06397) doi: 10.1007/s11263-018-1082-6

Mitchell, T. M. (1997). *Machine learning* (International ed., [Reprint.] ed.). New York, NY: McGraw-Hill. (OCLC: 846511832)

Movshovitz-Attias, Y., Kanade, T., & Sheikh, Y. (2016, March). How useful is photo-realistic rendering for visual learning? In *arXiv:1603.08152 [cs]*. Retrieved 2019-08-23, from http://arxiv.org/abs/1603.08152 (arXiv: 1603.08152)

Nguyen, V. N., Jenssen, R., & Roverso, D. (2018, July). Automatic autonomous vision-based power line inspection: A review of current status and the potential role of deep learning. *International Journal of Electrical Power & Energy Systems*, *99*, 107–120. Retrieved 2019-06-15, from http://www.sciencedirect.com/science/article/pii/S0142061517324444 doi: 10.1016/j.ijepes.2017.12.016

Nguyen, V. N., Jenssen, R., & Roverso, D. (2020, January). LS-Net: Fast Single-Shot Line-Segment Detector. *arXiv:1912.09532 [cs, eess]*. Retrieved 2020-06-02, from http://arxiv.org/abs/1912.09532 (arXiv: 1912.09532)

Prakash, A., Boochoon, S., Brophy, M., Acuna, D., Cameracci, E., State, G., ... Birchfield, S. (2018, October). Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data. In *2019 International Conference on Robotics and Automation (ICRA)*. Retrieved 2019-08-01, from http://arxiv.org/abs/1810.10093 (arXiv: 1810.10093)

Richardson, E., Sela, M., & Kimmel, R. (2016, September). 3D Face Reconstruction by

Learning from Synthetic Data. In *arXiv:1609.04387 [cs].* Retrieved 2019-08-22, from http://arxiv.org/abs/1609.04387 (arXiv: 1609.04387)

Simonyan, K., Vedaldi, A., & Zisserman, A. (2014, April). Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv:1312.6034 [cs]*. Retrieved 2019-12-05, from http://arxiv.org/abs/1312.6034 (arXiv: 1312.6034)

Smith, S. W. (1997). *The scientist and engineer's guide to digital signal processing* (1st ed ed.). San Diego, Calif: California Technical Pub.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014, January). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, *15*(1), 1929–1958. Retrieved 2019-10-09, from http://dl.acm.org/citation.cfm?id=2627435.2670313

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017, September). Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* Retrieved 2019-07-29, from http://arxiv.org/abs/1703.06907 (arXiv: 1703.06907)

Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., ... Birchfield, S. (2018, June). Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).* Retrieved 2019-07-17, from http://arxiv.org/abs/1804.06516 (arXiv: 1804.06516)

Varol, G., Romero, J., Martin, X., Mahmood, N., Black, M. J., Laptev, I., & Schmid, C. (2017). Learning From Synthetic Humans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 109–117). Retrieved 2019-08-23, from http://openaccess.thecvf.com/content_cvpr_2017/html/Varol_Learning_From_Synthetic_CVPR_2017_paper.html

Wilson, D. L., Coyle, J. R., & Thomas, E. A. (2017, November). Ensemble machine learning and forecasting can achieve 99% uptime for rural handpumps. *PLoS One; San Francisco*, *12*(11), e0188808. Retrieved 2019-11-20, from https://search.proquest.com/docview/1969919135/abstract/604C6A80855B4ACCPQ/1 doi: http://dx.doi.org/10.1371/journal.pone.0188808

Zhuang, P., Schwing, A. G., & Koyejo, S. (2019, July). FMRI data augmentation via synthesis. In *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019).* Retrieved 2019-07-18, from http://arxiv.org/abs/1907.06134 (arXiv: 1907.06134)

Appendix A

MobileNet's feature maps visualization

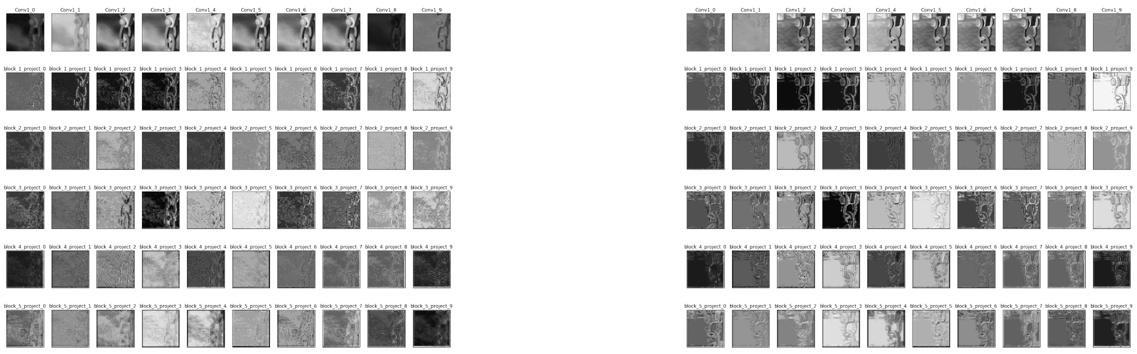


Figure A.1: MobileNet's feature maps visualization of sample 1a and 1b

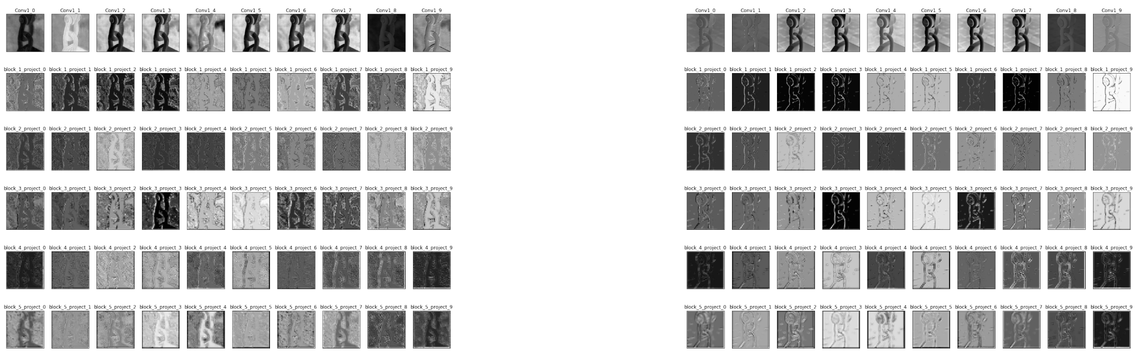


Figure A.2: MobileNet's feature maps visualization of sample 2a and 2b

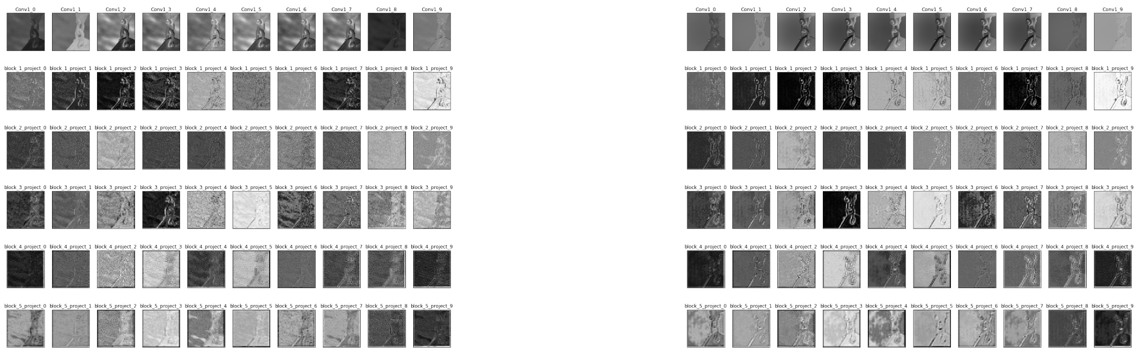


Figure A.3: MobileNet's feature maps visualization of sample 3a and 3b

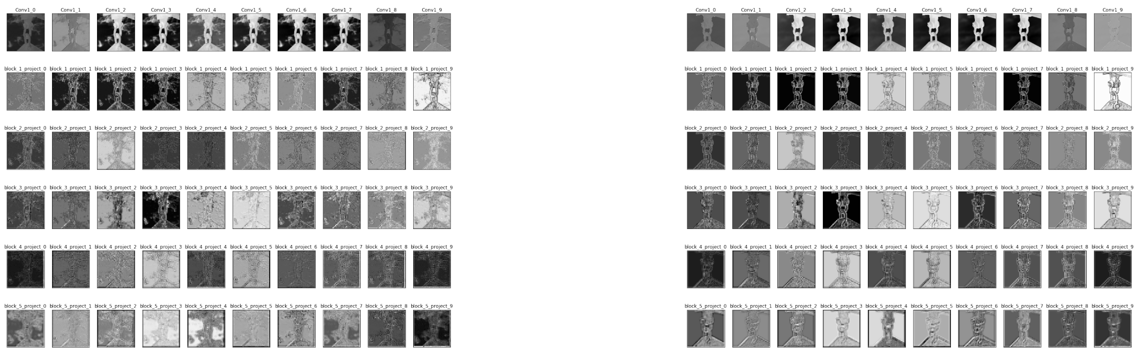


Figure A.4: MobileNet's feature maps visualization of sample 4a and 4b

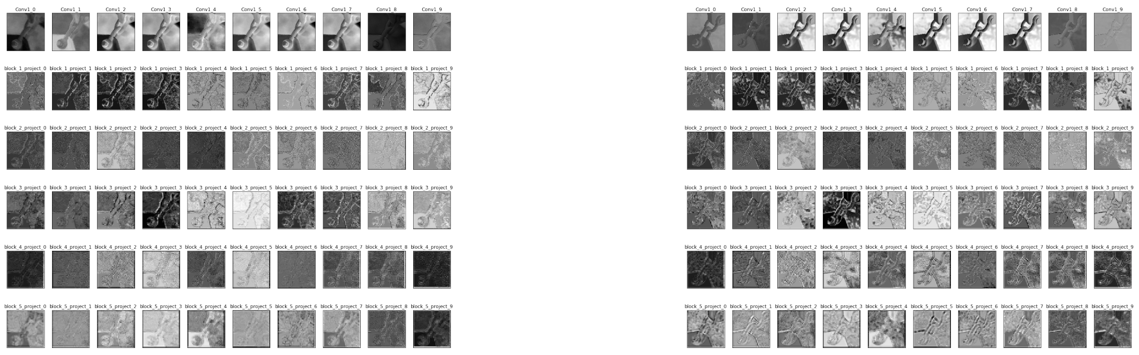


Figure A.5: MobileNet's feature maps visualization of sample 5a and 5b

Appendix B

ResNet50's feature maps visualization

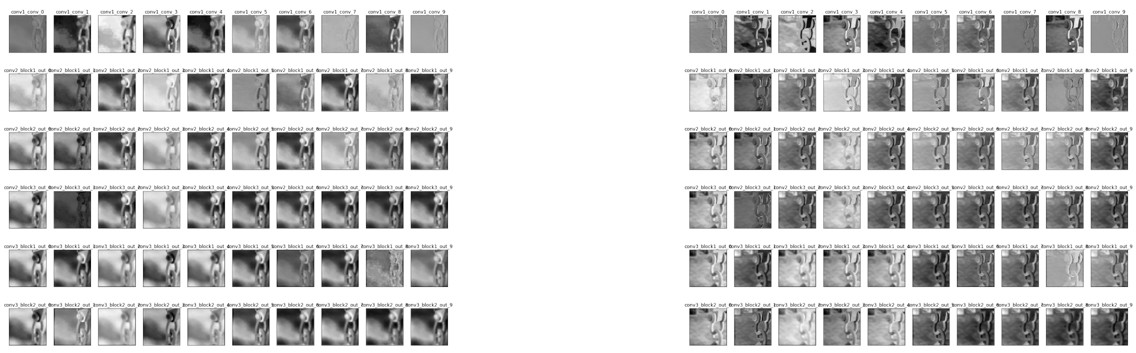


Figure B.1: ResNet50's feature maps visualization of sample 1a and 1b

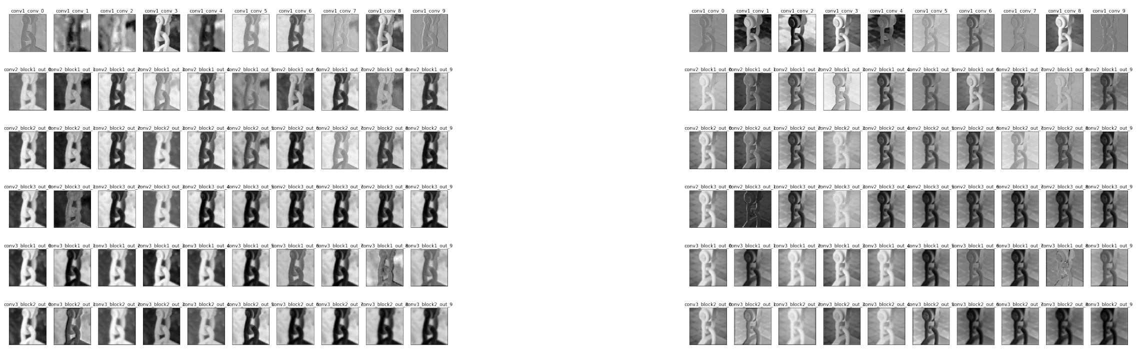


Figure B.2: ResNet50's feature maps visualization of sample 2a and 2b

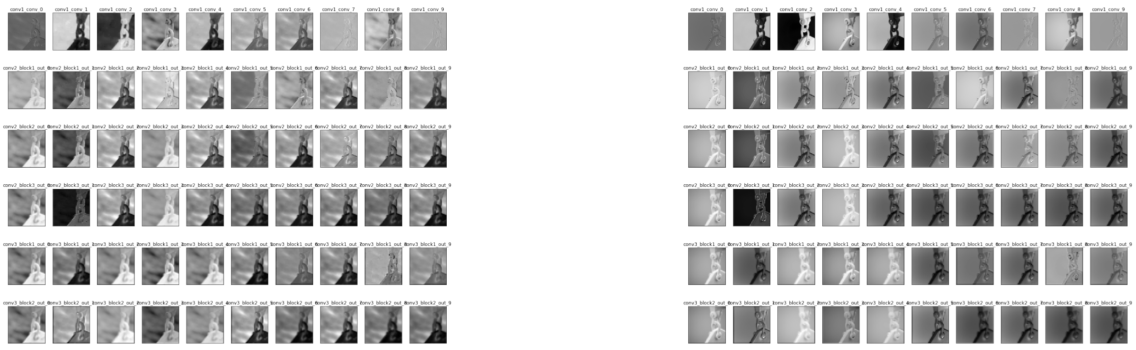


Figure B.3: ResNet50's feature maps visualization of sample 3a and 3b

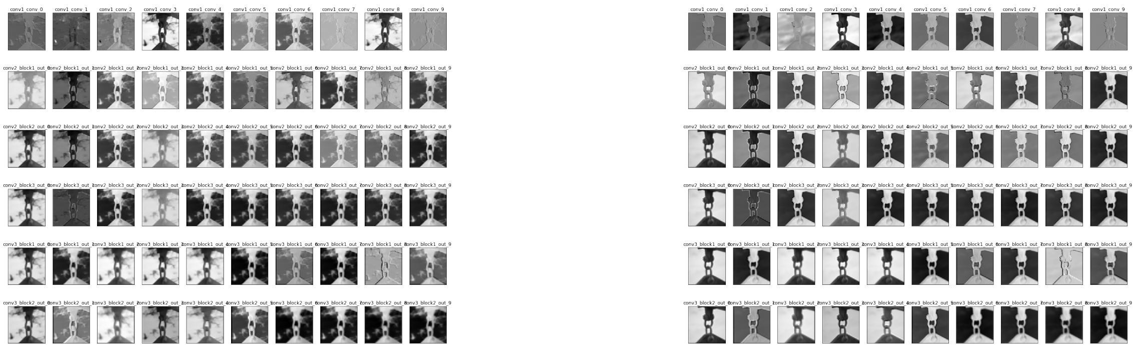


Figure B.4: ResNet50's feature maps visualization of sample 4a and 4b

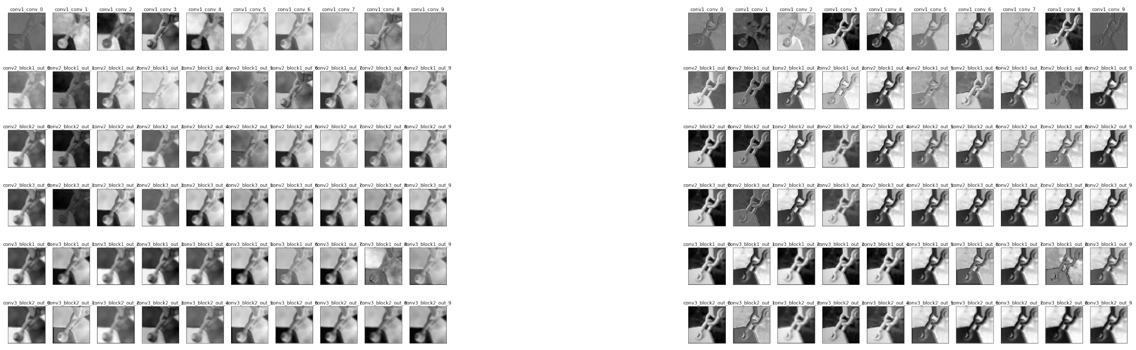


Figure B.5: ResNet50's feature maps visualization of sample 5a and 5b

Appendix C

Codes

C.1 Python sample code for training and testing

```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout,
    MaxPooling2D, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.framework import ops
from tensorflow.python.ops import math_ops
from tensorflow.python.keras import backend as K
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
    TensorBoard
import os
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


# In[2]:


batch_size = 8
IMG_HEIGHT = 512
IMG_WIDTH = 512
PATIENCE = 100
model_name = '08April_ResNet50_DataAug_0_AVGPooling.h5'


# In[3]:


# Specify GPU ID based on PCI ID
os.environ["CUDA_DEVICE_ORDER"]="PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"]="3"

```

```

# # Load the dataset from CSV file

# In[4]:


# Define generators for train, test and validation set. Only training data
    need data augmentation.
train_data_gen = ImageDataGenerator(rescale = 1./255,
                    rotation_range=45,
                    width_shift_range=.15,
                    height_shift_range=.15,
                    horizontal_flip=True)
test_data_gen = ImageDataGenerator(rescale = 1./255)
val_data_gen = ImageDataGenerator(rescale = 1./255)


# In[5]:


TRAIN_CSV = '/notebooks/two_cs_with_J/train_df.csv'
TEST_CSV = '/notebooks/two_cs_with_J/test_df.csv'
VAL_CSV = '/notebooks/two_cs_with_J/val_df.csv'
IMAGE_DIR = '/notebooks/two_cs_with_J/images'


# In[6]:



train_label_df = pd.read_csv(TRAIN_CSV)
test_label_df = pd.read_csv(TEST_CSV)
val_label_df = pd.read_csv(VAL_CSV)


# In[7]:


train_gen = train_data_gen.flow_from_dataframe(dataframe=train_label_df,
                                        directory = IMAGE_DIR,
                                        x_col = 'image_name',
                                        y_col = 'percentage',
                                        class_mode = 'raw',
                                        target_size = (IMG_HEIGHT,
    IMG_WIDTH),
                                        batch_size = batch_size)


# In[8]:


test_gen = test_data_gen.flow_from_dataframe(dataframe=test_label_df,
                                        directory = IMAGE_DIR,
                                        x_col = 'image_name',
                                        y_col = 'percentage',
                                        shuffe=False,
                                        class_mode = 'raw',
                                                   target_size = (IMG_HEIGHT,
    IMG_WIDTH),
                                                   batch_size = batch_size)


# In[9]:


val_gen = train_data_gen.flow_from_dataframe(dataframe=val_label_df,
                                                   directory = IMAGE_DIR,
                                                   x_col = 'image_name',
                                                   y_col = 'percentage',
                                                   class_mode = 'raw',
                                                   target_size = (IMG_HEIGHT,
    IMG_WIDTH),
                                                   batch_size = batch_size)


# # Define ResNet50 model with base network from TF HubApplcation

# In[10]:


base_model = tf.keras.applications.ResNet50V2(input_shape=(IMG_HEIGHT,
    IMG_WIDTH, 3), include_top=False)


# In[11]:


global_average_layer = tf.keras.layers.GlobalAveragePooling2D()


# In[12]:


output_layer = Dense(1, activation='linear')


# In[19]:


model_ResNet50v2 = Sequential([base_model, global_average_layer,
    output_layer])


# In[20]:


model_ResNet50v2.compile(optimizer='adam', loss=tf.losses.
    mean_absolute_error)


# # Train the model

# In[15]:


```

```
# calculate step/training_epoch and step/validation_epoch
steps_per_epoch = train_label_df.shape[0] // batch_size
validation_steps = val_label_df.shape[0] // batch_size


# In[17]:


# make directory for logs
logdir = os.path.join('/notebooks/models_2_way/logs', model_name)
os.mkdir(logdir)


# In[18]:


# early stopping, monitoring validation loss
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=
    PATIENCE, restore_best_weights=True)

# save best model
mc = ModelCheckpoint(model_name, monitor='val_loss', mode='min', verbose=1,
     save_best_only=True)

# tensorboard
tsb = TensorBoard(log_dir=logdir)


# In[ ]:


# Train the model and keep the history
history = model_ResNet50v2.fit_generator(
    train_gen,
    steps_per_epoch=steps_per_epoch,
    epochs=1000,
    validation_data=val_gen,
    validation_steps=validation_steps,
    callbacks = [es, mc, tsb]
)


# In[ ]:


# Visualize training curves via history
plt.rcParams['figure.figsize'] = (20, 20)
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.ylim([-0.05, 0.7])
plt.legend(loc='upper right')


# In[ ]:


```

```
# Evaluate the model on test set
model_ResNet50v2.evaluate(test_gen)


# In[ ]:


# Realese GPU
from numba import import cuda
cuda.select_device(0)
cuda.close()
```

Listing C.1: Code for training and evaluating on ResNet50

C.2 Feature maps visualization codes

```
layer_names = ['Conv1', 'block_1_project', 'block_2_project', '
    block_3_project', 'block_4_project', 'block_5_project']
def visualize_forward_pass(layer_names, pretrained_model, input_image):
    output_blocks = []
    for layer_name in layer_names:
        middle_model = Model(inputs=pretrained_model.input, outputs=
    pretrained_model.get_layer(layer_name).output)
        middle_model_output = middle_model.predict(input_image)
        output_blocks.append(middle_model_output)
    return output_blocks
```

Listing C.2: MobileNet's feature maps visualization code

C.3 Self-constructed ResNet50

```
def get_resnet():
    model = ResNetTypeII(layer_params=[3, 4, 6, 3])
    model.build(input_shape=(None, IMG_HEIGHT, IMG_WIDTH, 3))
    model.summary()
    return model
class ResNetTypeII(tf.keras.Model):
    def __init__(self, layer_params):
        super(ResNetTypeII, self).__init__()
        self.conv1 = tf.keras.layers.Conv2D(filters=64,
                                            kernel_size=(7, 7),
                                            strides=2,
                                            padding="same")
        self.bn1 = tf.keras.layers.BatchNormalization()
        self.pool1 = tf.keras.layers.MaxPool2D(pool_size=(3, 3),
                                               strides=2,
                                               padding="same")

        self.layer1 = make_bottleneck_layer(filter_num=64,
                                            blocks=layer_params[0])
        self.layer2 = make_bottleneck_layer(filter_num=128,
                                            blocks=layer_params[1],
                                            stride=2)
        self.layer3 = make_bottleneck_layer(filter_num=256,
                                              blocks=layer_params[2],
                                              stride=2)
        self.layer4 = make_bottleneck_layer(filter_num=512,
                                              blocks=layer_params[3],
                                              stride=2)

        self.avgpool = tf.keras.layers.GlobalAveragePooling2D()
        self.reg = tf.keras.layers.Dense(units=1, activation='linear')

    def call(self, inputs, training=None, mask=None):
        x = self.conv1(inputs)
        x = self.bn1(x, training=training)
        x = tf.nn.relu(x)
        x = self.pool1(x)
        x = self.layer1(x, training=training)
        x = self.layer2(x, training=training)
        x = self.layer3(x, training=training)
        x = self.layer4(x, training=training)
        x = self.avgpool(x)
        output = self.reg(x)

        return output

def make_bottleneck_layer(filter_num, blocks, stride=1):
    res_block = tf.keras.Sequential()
    res_block.add(BottleNeck(filter_num, stride=stride))

    for _ in range(1, blocks):
        res_block.add(BottleNeck(filter_num, stride=1))

    return res_block

class BottleNeck(tf.keras.layers.Layer):
    def __init__(self, filter_num, stride=1):
        super(BottleNeck, self).__init__()
        self.conv1 = tf.keras.layers.Conv2D(filters=filter_num,
                                            kernel_size=(1, 1),
                                            strides=1,
                                            padding='same')
        self.bn1 = tf.keras.layers.BatchNormalization()
        self.conv2 = tf.keras.layers.Conv2D(filters=filter_num,
                                            kernel_size=(3, 3),
                                            strides=stride,
                                            padding='same')
        self.bn2 = tf.keras.layers.BatchNormalization()
        self.conv3 = tf.keras.layers.Conv2D(filters=filter_num * 4,
                                            kernel_size=(1, 1),
                                            strides=1,
                                            padding='same')
        self.bn3 = tf.keras.layers.BatchNormalization()

        self.downsample = tf.keras.Sequential()
        self.downsample.add(tf.keras.layers.Conv2D(filters=filter_num * 4,
                                                   kernel_size=(1, 1),
                                                   strides=stride))
        self.downsample.add(tf.keras.layers.BatchNormalization())

    def call(self, inputs, training=None, **kwargs):
        residual = self.downsample(inputs)

        x = self.conv1(inputs)
        x = self.bn1(x, training=training)
        x = tf.nn.relu(x)
        x = self.conv2(x)
        x = self.bn2(x, training=training)
        x = tf.nn.relu(x)
        x = self.conv3(x)
        x = self.bn3(x, training=training)

        output = tf.nn.relu(tf.keras.layers.add([residual, x]))

        return output
```

Listing C.3: Self-constructed ResNet50 based on Keras layers